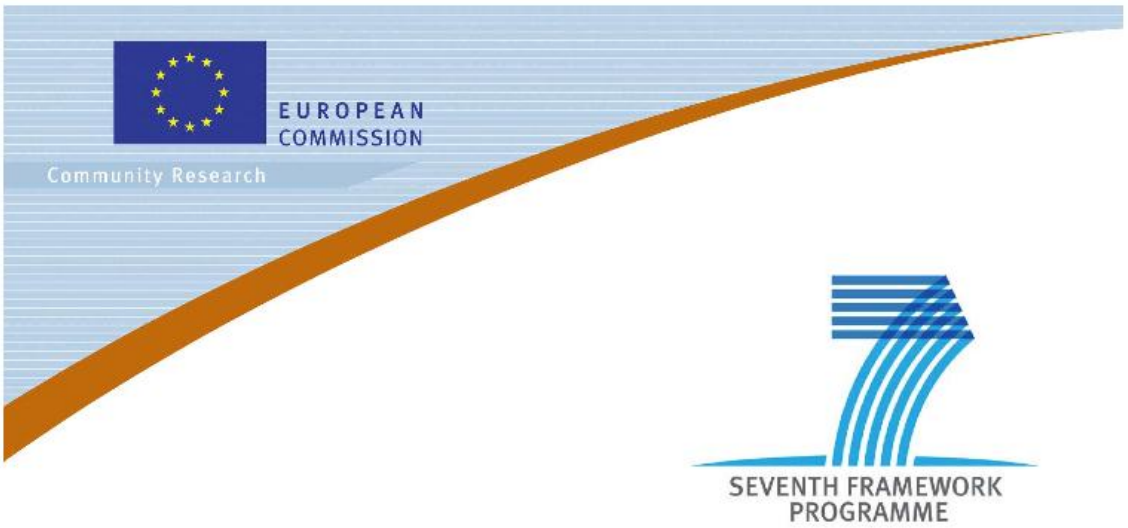


Private Public Partnership Project (PPP)

Large-scale Integrated Project (IP)

D.7.4.3: FI-WARE User and Programmers Guide

Project acronym: FI-WARE
Project full title: Future Internet Core Platform
Contract No.: 285248
Strategic Objective: FI.ICT-2011.1.7 Technology foundation: Future Internet Core Platform
Project Document Number: ICT-2011-FI-285248-WP7-D.7.4.3
Project Document Date: 2014-07-10
Deliverable Type and Security: Public
Author: FI-WARE Consortium
Contributors: FI-WARE Consortium

1.1 Executive Summary

This document describes the usage of each Generic Enabler provided by the "Interfaces to Network and Devices" (I2ND) chapter. The document also details the required software elements and the procedure to setup an environment for the development of applications that can exploit the capabilities of the I2ND Generic Enablers.

1.2 About This Document

This document comes along with the Software implementation of components, each release of the document being referred to the corresponding Software release (as per D.x.3), to provide documentation of the features offered by the components and interfaces to users/adopters. Moreover, it explains the way they can be exploited in their developments.

1.3 Intended Audience

The document targets users as well as programmers of FI-WARE Generic Enablers.

1.4 Chapter Context

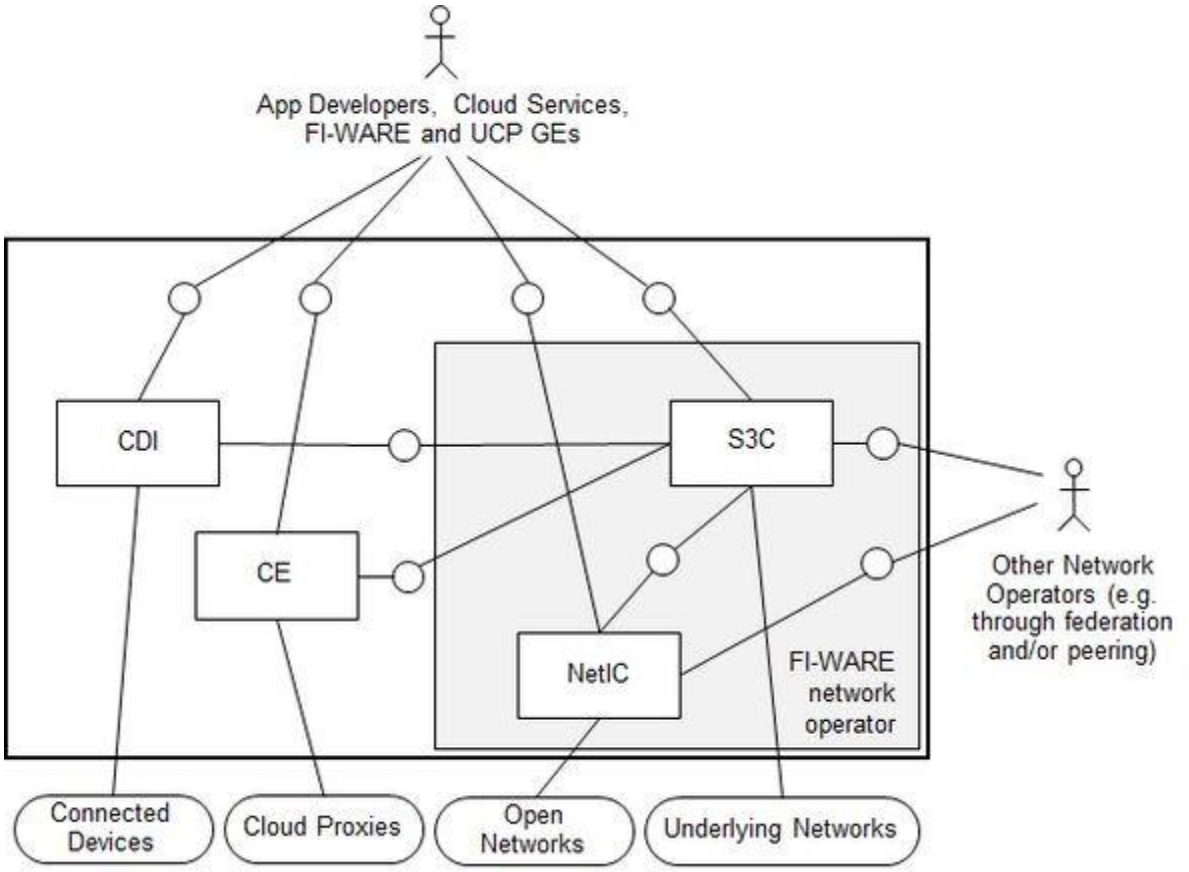
The I2ND chapter defines an enabler space for providing Generic Enablers (GEs) to run an open and standardised network infrastructure. The infrastructure deals with highly sophisticated terminals, as well as with highly sophisticated proxies, on one side, and with the network operator infrastructure on the other side. This latter will be implemented by physical network nodes, which typically will be under direct control of an operator, or the node functionality will be virtualised – in this case the I2ND functionality can be accessed by further potential providers, like virtual operators. The I2ND chapter defines four GEs, as represented in the figure:

- CDI (Connected Device Interface) towards the Connected Devices. These devices include, but are not limited to, mobile terminals, tablets, set top boxes and media phones.
- CE (Cloud Edge) towards the Cloud Proxies. Cloud Proxies are gateways, which will connect and control a set-up of nodes towards the Internet or/and an operator network.
- NetIC (Network Information and Control) towards Open Networks. Open Networks are following the idea of flow based controlled networks, and can be used for virtualisation of networks.
- S3C (Service Capability, Connectivity and Control) towards Underlying Networks. The underlying networks are following standards such as Next Generation Networks (NGNs) or Next Generation Mobile Networks (NGMNs). In the case of the S3C specified in I2ND the baseline underlying network will be the Evolved Packet Core (EPC) by 3GPP.

Each of the GEs of I2ND have specific interfaces which can be accessed by Application Developers, Cloud Services, FI-WARE and third party Enablers. Besides typical interfaces to functionality offered by suche GEs, it is worth mentioning that:

- The GE S3C is the central point of the I2ND architecture. I2ND develops an enabling environment which can be used by network operators. Together with NetIC, both GEs build the environment of an operator, which might even be a virtual operator. S3C can be seen as the GE to run and steer the network infrastructure.
- The interfacing between S3C and CDI provides status and control information exchange of the device and remote control capabilities.

- Cloud proxies can be part of an operator infrastructure. Therefore it is necessary to have access to these network nodes through a standardised interface.



1.5 Structure of this Document

The document is generated out of a set of documents provided in the public FI-WARE wiki. For the current version of the documents, please visit the public wiki at http://wiki.fi-ware.org/

The following resources were used to generate this document:

D.7.4.3 User and Programmers Guide front page

Connected Device Interfacing - User and Programmers Guide

Cloud Edge - User and Programmers Guide

Cloud Edge OSGi - User and Programmers Guide

NetIC GE - OFNIC - User and Programmers Guide

NetIC GE - VNP - User and Programmers Guide

NetIC GE - VNEIC - User and Programmers Guide

NetIC GE - Altoclient - User and Programmers Guide

S3C GE - API Mediation - User and Programmers Guide

S3C GE - Telecom AS - User and Programmers Guide

S3C GE - Network Identity Management - User and Programmers Guide

S3C GE - Seamless Network Connectivity - User and Programmers Guide

S3C GE - EPC OTT API - User and Programmers Guide

S3C GE - Network Positioning Enabler - User and Programmers Guide

1.6 Typographical Conventions

Starting with October 2012 the FI-WARE project improved the quality and streamlined the submission process for deliverables, generated out of our wikis. The project is currently working on the migration of as many deliverables as possible towards the new system.

This document is rendered with semi-automatic scripts out of a MediaWiki system operated by the FI-WARE consortium.

1.6.1 Links within this document

The links within this document point towards the wiki where the content was rendered from. You can browse these links in order to find the "current" status of the particular content.

Due to technical reasons part of the links contained in the deliverables generated from wiki pages cannot be rendered to fully working links. This happens for instance when a wiki page references a section within the same wiki page (but there are other cases). In such scenarios we preserve a link for readability purposes but this points to an explanatory page, not the original target page.

In such cases where you find links that do not actually point to the original location, we encourage you to visit the source pages to get all the source information in its original form. Most of the links are however correct and this impacts a small fraction of those in our deliverables.

1.6.2 Figures

Figures are mainly inserted within the wiki as the following one:

```
[[Image:....|size|alignment|Caption]]
```

Only if the wiki-page uses this format, the related caption is applied on the printed document. As currently this format is not used consistently within the wiki, please understand that the rendered pages have different caption layouts and different caption formats in general. Due to technical reasons the caption can't be numbered automatically.

1.6.3 Sample software code

Sample API-calls may be inserted like the following one.

```
http://[SERVER_URL]?filter=name:Simth*&index=20&limit=10
```

1.7 Acknowledgements

The following partners contributed to this deliverable: ALU, DT, FRAUNHOFER, FT, INTEL, NSN, TRDF, TI, UNIROMA1

1.8 Keyword list

FI-WARE, PPP, Architecture Board, Steering Board, Roadmap, Reference Architecture, Generic Enabler, Open Specifications, I2ND, Cloud, IoT, Data/Context Management, Applications/Services Ecosystem, Delivery Framework , Security, Developers Community and Tools , ICT, es.Internet, Latin American Platforms, Cloud Edge, Cloud Proxy.

1.9 Changes History

Release	Major changes description	Date	Editor
v1	First draft of deliverable submission generated	2014-07-10	TI
V2	Revised version for final delivery	2014-07-11	TI

1.10 Table of Contents

1.1 Executive Summary 2
1.2 About This Document 3
1.3 Intended Audience 3
1.4 Chapter Context 3
1.5 Structure of this Document 4
1.6 Typographical Conventions 5
1.6.1 Links within this document 5
1.6.2 Figures 5
1.6.3 Sample software code 6

1.7 Acknowledgements 6
1.8 Keyword list 6
1.9 Changes History 6
1.10 Table of Contents 6
2 Connected Device Interfacing - User and Programmers Guide 10
2.1 Introduction 10
2.2 Users Guide 10
2.2.1 A-CDI GEi architecture 10
2.2.2 Software and Hardware environment 11
2.3 Programmers Guide 12
2.3.1 Including Libraries 12
2.3.2 Library Initialization 13
2.3.3 CDI GEi API Discovery 13
2.3.4 CDI GEi API example usage 14
3 Cloud Edge - User and Programmers Guide 37
3.1 Introduction 37
3.1.1 Goal of this document 37
3.2 User Guide 38
3.3 Programmers Guide 38
3.3.1 Create a CloudEdge Virtual Machine image for STB (HTTP web server) 38
3.3.2 Run a CloudEdge Virtual Machine image on STB (HTTP web server) 40
4 Cloud Edge OSGi - User and Programmers Guide 41
4.1 Introduction 41
4.2 User Guide 41
4.3 Programmers Guide 41
4.3.1 Check the OSGi properties 42
5 NetIC GE - OFNIC - User and Programmers Guide 44
5.1 Introduction 44
5.2 User Guide 44
5.2.1 OFNIC GUI 44
5.3 Programmer Guide 46
5.3.1 Navigation with browser 46
5.3.2 Apache HttpClient 49
5.3.3 How to extend OFNIC GEi 50

6 NetIC GE - VNP - User and Programmers Guide 51
6.1.1 Introduction 51
6.1.2 User Guide 52
6.1.3 Programmer Guide 54
7 NetIC GE - VNEIC - User and Programmers Guide 56
7.1.1 Introduction 56
7.1.2 User Guide 57
7.1.3 Programmer's Guide 57
8 NetIC GE - Altoclient - User and Programmers Guide 62
8.1.1 Introduction 62
8.1.2 User Guide 62
8.1.3 Programmer's Guide 63
9 S3C GE - API Mediation - User and Programmers Guide 70
9.1 Introduction 70
9.1.1 What is Open Service Access? 70
9.2 User guide 70
9.2.1 What does Open Service Access? 70
9.2.2 How does Open Service Access works? 71
9.2.3 Authorization model 72
9.2.4 Submitting a service 72
9.3 Programmer guide 74
9.3.1 Typical Use Case 74
9.3.2 Ports in use after installation 76
9.3.3 Web Services 76
10 S3C GE - Telecom AS - User and Programmers Guide 84
10.1 Introduction 84
10.2 User Guide 84
10.3 Programmers guide 84
10.3.1 Samples of enabler using of mainly requests 84
10.3.2 Constraint of usage 87
11 S3C GE - Network Identity Management - User and Programmers Guide 88
11.1 Introduction 88
11.2 Users Guide 88
11.3 Programmers Guide 88

11.3.1 Preconditions ... 88
11.3.2 Device Capabilities ... 89
11.3.3 Third Party Call with device specific addressing ... 91
12 S3C GE - Seamless Network Connectivity - User and Programmers Guide ... 93
12.1 Introduction ... 93
12.2 User Guide ... 93
12.2.1 Flow setup ... 93
12.3 Programmers Guide ... 96
13 S3C GE - EPC OTT API - User and Programmers Guide ... 97
13.1 Introduction ... 97
13.2 User Guide ... 98
13.3 Programmer Guide ... 98
13.3.1 Starting a QoS Session ... 98
13.3.2 Modification of Reserved Resources ... 100
13.3.3 Ending a QoS Session ... 100
13.3.4 Getting the QoS Status of a User ... 101
13.3.5 Subscribing for Event Notfications ... 101
13.3.6 Transmission of an IP message ... 101
14 S3C GE - Network Positioning Enabler - User and Programmers Guide ... 102
14.1 S3C GE - Network Positioning Enabler - User and Programmers Guide ... 102
14.1.1 Introduction ... 102
14.2 User Guide ... 103
14.2.1 Client ... 103
14.2.2 3rd party interface ... 106
14.3 Programmers Guide ... 108
14.3.1 Registering and authenticating using OpenId ... 108
14.3.2 Logout ... 108
14.3.3 Getting the location of a user from the ANDSF ... 109
14.3.4 Subscription and Unsubscription ... 112
14.3.5 Manipulation of a notification ... 112
14.3.6 Position request ... 114
14.3.7 Sending a push notification ... 115

2 Connected Device Interfacing - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

2.1 Introduction

This document provides a useful guide for developers and users of web applications written on top of the A-CDI Generic Enabler implementation of the Connected Device Interfacing Open Specifications API. Specific sections are dedicated to developer tools intended to help them during development, test and deployment of CDI GEi based Webapps. These include illustrations demonstrating how to discover and use the A-CDI GEi APIs.

CDI GEi installation and administration information can be found at the Connected Device Interfacing - Installation and Administration Guide page.

2.2 Users Guide

CDI defines a set of common APIs, designed for developers to extend the potentiality of web applications, and providing application portability. The CDI GE implementation, A-CDI, is designed as a JavaScript library with additional native software components. A-CDI APIs are defined in a JavaScript library `cdi.js` and can be used either by itself inside a normal web page, or as Widgets that can be run from the CDI-Webinos Android application.

A-CDI GEi provides a wide range of functionality via a big API. The API is broken into functional blocks. A Functional block is a group of APIs which provide a given set of functionality, e.g. Geolocation would be one functional block, and File System Access would be another. A-CDI GEi can be deployed on a wide range of devices. Each device offers different functionality, not all functional blocks may be available on all devices, therefore A-CDI provides a method of detecting the presence of a functional block. A complete list of all of the available functional blocks can be found in the FI-WARE Architecture Description. When the `cdi.js` is used without the native software components it provides less functional blocks than would otherwise be available.

2.2.1 A-CDI GEi architecture

The current FI-WARE release of A-CDI is composed of two independent subsystems:

- CDI-Webinos platform, which supports Android devices.
- Distribute Compute Framework, also called Sisyphus. It is available as a Javascript library for client and server architectures.

CDI-Webinos is implemented on top of Webinos, a platform developed within an EU FP7 Project. The native software components are implemented as Webinos extensions for the Android operating system, and provide functionality which requires access to native, or host operating system interfaces. Sisyphus is a pure Javascript library and only require a system able to parse and execute this language, like any recent web browser (such as Firefox, Opera, Chrome) or a javascript runtime module (such as the nodejs platform). A-CDI provides a single JavaScript object `$cdi` through which all functional blocks and their APIs are available.

2.2.2 Software and Hardware environment

2.2.2.1 *CDI-Webinos Platform (Android Client)*

CDI-Webinos is a client only platform which supports Android devices (with minimum version Android 3.x, Android 4.x preferred), and expose a sets of APIs available in Javascript. These APIs can be called and executed in a HTML/JS code opened in any recent browsers (Firefox, Opera, Chrome) or as part of a Widget (a zipped archive containing HTML/JS) launched from the CDI-Webinos platform. To use the CDI-Webinos platform, it is sufficient to add few lines of Javascript code to include its libraries and call an initialization procedure.

2.2.2.2 *Distributed Compute Framework (Sisyphus Server)*

The cloud (server) requirements for a Sisyphus Framework is as follows:

- RAM: 512 megs
- CPU: 1 core
- Hard Drive: 8 Gig (used 1.7G, including development tools not needed for a simply deployment)

Sisyphus and the example application are node.js scripts which requires the following software components:

- Ubuntu 12.04 LTS (32bit)

- Node.js
- Socket.io
- Socket.io-client
- The "q" promise framework
- jQuery (needed for development & example code)
- Express (needed for development & example code)

2.3 Programmers Guide

CDI GEi makes it possible for a programmer to create a web page with rich functionality and deploy it as a hosted CDI-Webinos application with enhanced functionality. To do this all CDI GEi based applications must be structured in the same way. Firstly the programmer must include the CDI-Webinos (which includes CDI GEi libraries), then wait for the CDI GEi Library to Initialize, lastly the programmer must validate that the functional block containing the functionality they wish to use is available. These three steps are covered in detail below.

A CDI GEi based application can be developed and deployed in 3 ways:

1. *As an HTML document (with no native software)*: The application is a HTML5 web page which does not use any of the enhanced native features of CDI GEi.
2. *As an HTML document (with native software)*: A web page which connects to and uses a CDI-Webinos core located on the viewing device. Here the application exploits the CDI-Webinos messaging system to convey JavaScript calls out of the browser, toward the CDI-Webinos PZP (the Connected Device Interfacing - Installation and Administration Guide provides the instructions on how to install and run the PZP). This is achieved by using the HTML5 WebSockets API, that implies the Browser must support the W3C WebSockets specification. WebSockets is fully supported by Chrome and Firefox, however you can refer this page to check WebSockets compliancy within your browser.
3. *As a CDI-Webinos widget*: If the application is packaged as a CDI-Webinos widget, it will be executed in the WebRT provided by CDI-Webinos. CDI-Webinos widgets are installed by opening the CDI-Webinos Application(that comes with the apks installation) and scanning the SD-Card of your device looking for **.wgt* files. A specific section is dedicated to the packaging and installation of CDI-Webinos widgets.

2.3.1 Including Libraries

CDI GEi based applications are web applications, and are therefore HTML documents. To use CDI GEi with the native software components it is necessary to import the `webinos.js` file and the `cdi.js` files. It is important that they are imported *in that order (`webinos.js<code> THEN <code>cdi.js`).*

```
<html>
```

```
    <head>

        <!-- Note: To use this code both as a webpage and widget, add
both webinos.js inclusion lines below, respecting the same order  -->

        <script type="text/javascript" src="/static/webinos.js"></script>

        <script type="text/javascript"
src="http://localhost:8080/webinos.js"></script>

        <script type='text/javascript' src='cdi.js'></script>

        <script type='text/javascript'>$cdi.init();</script> <!-- CDI
initialisation starts here -->

        <!-- ... -->

    </head>

    <body>

        <!-- body here -->

    </body>

</html>
```

NOTE: After including the CDI library the developer should invoke the `$cdi.init();` function. This tell's CDI to start-up, it will then watch for CDI-Webinos and/or jQuery start-up events before initialising its self. Once initialised CDI will emit an `onCDIReady` event. See below for more details.

2.3.2 Library Initialization

The CDI GEi framework performs a number of initialization steps when it loads and is not immediately available for use. Any code which wishes to make use of the CDI GEi API must wait until it is available. This is possible by waiting for the `onCDIReady` event:

```
      window.addEventListener("onCDIReady", function(evt) {

        // ... make use of CDI APIs here

      }, false);
```

This event always happens after the document is ready and has been loaded. It therefore supersedes these existing DOM events.

2.3.3 CDI GEi API Discovery

CDI GEi can work across a number of different types of devices and each different device can offer a different set of functionality. To cope with this CDI GEi APIs are broken down into functional blocks. It is

possible to query to see if a device supports a specific functional block. The code example below shows how to check for the presence of the QoE and Geolocation APIs:

```
window.addEventListener("onCDIReady", function(evt) {
    if ( $cdi.qoe.availability == true ) {
    // .. use QoE API here
    }

    if ( $cdi.sensors.availability.geolocation == true ) {
        // ... use the Geolocation API here
    }

}, false);




}, false);
```

More detailed info about the API are available in the CDI Open Specification document.

2.3.4 CDI GEi API example usage

This section is aimed at providing a complete example of a Webapp based on CDI GEi. There are two examples, one focused on the QoE functionality, and the other demonstrating a simple Geolocation use case.

Note the following examples use the jQuery library.
Developers are then supposed to import jQuery before executing the code.

2.3.4.1 *Basic Web Application : QoE*

The tutorial will show how the QoE API can be used to control the user's experience for a video stream application. This example shows how the QoE API works for a video streaming application that wishes to offer QoE support in its UI. The following code can be run on top the local web Browser available in your device, and relies on a running PZP instance equipped with the FI-WARE QoE API Handler module.

Please be aware that QoE functionality is only available when creating an A-CDI application which uses native software functionality. QoE only works when A-CDI runs with CDI-Webinos, which embeds the QoE Android Java code and makes it accessible to Javascript as a CDI-Webinos exposed API.

The following html file will be named as `cdi-qoe-helloworld.html`.

```
<html lang="en">

<head>

  <meta charset='utf-8'>

  <title>HTML5 Video and QoE API</title>

        <!-- Note: To use this code both as a webpage and widget, add
both webinos.js inclusion lines below, respecting the same order  -->

        <script type="text/javascript" src="/static/webinos.js"></script>

        <script type="text/javascript" charset="utf-8"
src="http://localhost:8080/webinos.js"></script>

        <script type="text/javascript" charset="utf-8"
src="cdi.js"></script>

        <script type="text/javascript">

                              var media_events = new Array();

                      media_events["play"] = 0;

                      media_events["pause"] = 0;

                      media_events["ended"] = 0;


                      var ctxtTimerTask;

                      var flowid = 0;

                      var flowurl;


                      /*Called when the document is loaded*/

                      function init() {


       $cdi.qoe.setupNetworkController($cdi.qoe.NetworkControllerType.NON
E,     "",

                              function(msg) {


                              document._video =
document.getElementById("video");
```

```
                                flowurl =
document.getElementById('mp4').getAttribute('src');

                                //get the video URL

                                $cdi.qoe.bindFlow(flowurl, //bind the video
URL to the QoE framework

                                function(msg) {

                                        flowid = msg.flowID;


       document.getElementById("qoe_flowid").innerHTML = "Assigned flow
ID: " + flowid



                                }, function(errmsg) {



                                        // If the bind exists (EXISTING_BIND =
5) go on

                                        if (errmsg.bcerr.code =
$cdi.qoe.ErrorCode.EXISTING_BIND) {

                                                flowid = errmsg.flowID;


       document.getElementById("qoe_flowid").innerHTML = "Assigned flow
ID: " + flowid

                                        }

                                        alert("An error occurred while binding
the flow: " + '\n' + errmsg.bcerr.code_desc);

                                });


                                init_events();




                                }, function(errmsg) {



                                        alert("An error occurred while
configuring the controller: " + '\n' + errmsg.bcerr.message);
```

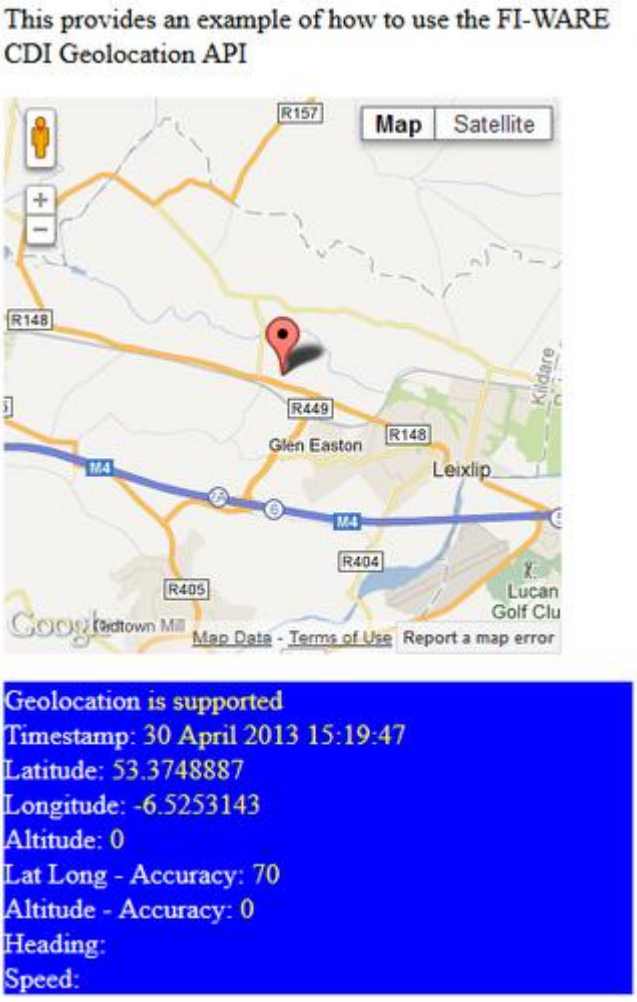
```
                        });


                }


                function init_events() {
                        for (key in media_events) {//Register
listeners for video events
                                document._video.addEventListener(key,
capture, false);
                        }
                }



                function startCtxtTimedTask(){
                        ctxtTimerTask = setInterval(getContextInfo,
1000);
                }


                function clearCtxtTimedTask(){
                        clearInterval(ctxtTimerTask);
                }



                window.addEventListener("onCDIReady", init, false);


                /*Called to update the UI with fresh context info*/
                function getContextInfo() {
                        $cdi.qoe.getContext(flowid, function(msg) {
```

```
	document.getElementById("qoe_status").innerHTML = "Monitoring
status: " + msg.ctxtdump.monitor;


	document.getElementById("qoe_feedback").innerHTML = "Click Rate: "
+ msg.ctxtdump.CR;


	document.getElementById("qoe_bandwidth").innerHTML = "Current
bandwidth: " + msg.ctxtdump.BW + " kbps";
                        }, function(errmsg) {
                        });


                }


                /*Handle video events*/
                function capture(event) {
                        if (event.type == "play") {
                                $cdi.qoe.runQoeMonitor(flowid,
startCtxtTimedTask, function(errmsg) {
                                });
                        } else if (event.type == "pause") {
                                $cdi.qoe.pauseQoeMonitor(flowid,
clearCtxtTimedTask, function(errmsg) {
                                });
                        } else if (event.type == "ended") {
                                clearCtxtTimedTask();
                        }
                }


                /*Attached to the QoE feedback button*/
                var feedback = function() {
```

```
                                $cdi.qoe.giveFeedback(flowid, getContextInfo,
function(errmsg) {

                                });

                        }

  </script>

</head>

<body>

  <h1>QoE API and HTML5 video example</h1>

  <p>This page is a demonstrations of the QoE API applied to a video
streaming application. When the video is played,

        its URL is bound to the QoE monitoring system, and a numeric
FlowID is generated.

        A button to provide bad QoE feedbacks is provided, together with
fields to track a subset of the flow's context</p>

  <div>

       <video id='video'

      controls preload='none'

      poster="http://media.w3.org/2010/05/sintel/poster.png">

      <source id='mp4'

        src="http://media.w3.org/2010/05/sintel/trailer.mp4"

        type='video/mp4'>

      <source id='webm'

        src="http://media.w3.org/2010/05/sintel/trailer.webm"

        type='video/webm'>

      <source id='ogv'

        src="http://media.w3.org/2010/05/sintel/trailer.ogv"

        type='video/ogg'>

      <p>Your user agent does not support the HTML5 Video element.</p>

       </video>

       <div>
```

```
                <input class="left" type='button' onclick='feedback()'
value='give feedback!'/>

                <div class="right" id="qoe_feedback">Click Rate: 0.0</div>

                <div class="right" id="qoe_flowid">Flow ID: undefined</div>

                <div class="right" id="qoe_status">Monitoring status:
UNKNOWN (not started yet)</div>

        </div>

   </div>

</body>

</html>
```

2.3.4.2 *Basic Web Application : GeoLocation*



The example in the picture shows how the Geolocation API works. It uses A-CDI GEi to obtain the geographic location of the device, then plots this on a Google Map. It provides full diagnostic information, displaying all the information returned to it by A-CDI. The following code can be run on top the local web

Browser available in your device, and relies on a running PZP instance equipped with the FI-WARE QoE API Handler module. The following html file will be named as `cdi-geolocation-helloworld.html`.

(1)Including the Libraries

Firstly we must include the correct libraries. In this example application we're also using the jQuery library, but JQuery is not compulsory. It is possible to write this application without jQuery, the author has used it out of convenience. This example uses the Google Map tool to plot the current location of the device, and therefore it also includes the Google Map's API.

```
<html>

    <meta name="viewport" content="initial-scale = 1.0,maximum-scale =
1.0" />

  <head>

    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />

    <script type="text/javascript" src="jquery-1.9.1.js"></script>

    <!-- Note: To use this code both as a webpage and widget, add both
webinos.js -->

    <!-- inclusion lines below, in the same order  -->

    <script type="text/javascript" src="/static/webinos.js"></script>

    <script type="text/javascript"
src="http://localhost:8080/webinos.js"></script>

    <script type="text/javascript" src="cdi.js"></script>

    <script
src="https://maps.googleapis.com/maps/api/js?v=3.exp&sensor=true"></scrip
t>

  <!-- ... -->

  </head>
```

(2)Library Initialization

Before using the A-CDI API it is necessary to wait for it to Initialize. This is achieved by waiting for the "`onCDIReady`" event.

```
        window.addEventListener("onCDIReady", function(evt) {

        /* ... */

        }, false);
```

(3)CDI GEi API Discovery

Once initialized it is necessary to confirm that the platform the application is running on supports geolocation. This is done by checking the `$cdi.sensors.availability.geolocation` boolean value. True indicates that the geolocation functional block is present.

```
        window.addEventListener("onCDIReady", function(evt) {
            if ( $cdi.sensors.availability.geolocation ) {
                $("#geo-support").html("is supported");
                startGeoLocationWatching();
            } else {
                /** geolocation is not available **/
            }
        }, false);
```

Once the functional block is present the code invokes the `startGeoLocationWatching()` function.

(4)Using the CDI GEi API (Geolocation)

The `startGeoLocationWatching()` contains all the code which uses the geolocation API. This functional initially sets up the Google Map object before calling the A-CDI Geolocation functionality.

```
        function startGeoLocationWatching() {
           // init the Google map object here.
            var mapOptions = {
                zoom: 12,
                mapTypeId: google.maps.MapTypeId.ROADMAP
            };
            map = new
google.maps.Map(document.getElementById('googlemap'), mapOptions);

            //request the current position

$cdi.sensors.geolocation.getCurrentPosition(function(position){
                geolocationPositionCallBack(position);
                geolocationWatchId =
$cdi.sensors.geolocation.watchPosition(geolocationPositionCallBack,
geolocationPositionCallBackError,
```

```
                {
                    enableHighAccuracy: true,
                    timeout: 60000,
                    maximumAge: 60000
                });
            }, geolocationPositionCallBackError,
            {
                enableHighAccuracy: true,
                timeout: 60000,
                maximumAge: 60000
            });
        };
```

The geolocation functional block is available on the `$cdi.sensors.geolocation` object. The code above requests the current position, if this request completes successfully the lambda function is invoked and map is updated by calling the application function `geolocationPositionCallBack` and passing the `position` object supplied by the A-CDI API. After this the code calls `watchPosition` to register for any changes to the devices location. From this point onwards, every time the device's position changes, `geolocationPositionCallBack` will be invoked.

(5)Handling Geolocation Data

The `geolocationPositionCallBack` function deals with the geolocation data when it is supplied by A-CDI. It received the `position` object as supplied by the A-CDI API. The `position` object contains more than just the latitude and longitude information. It can also include information on speed, if the device is traveling, accuracy information and altitude. The function uses some jQuery calls to update fields on the page page with all the information returned.

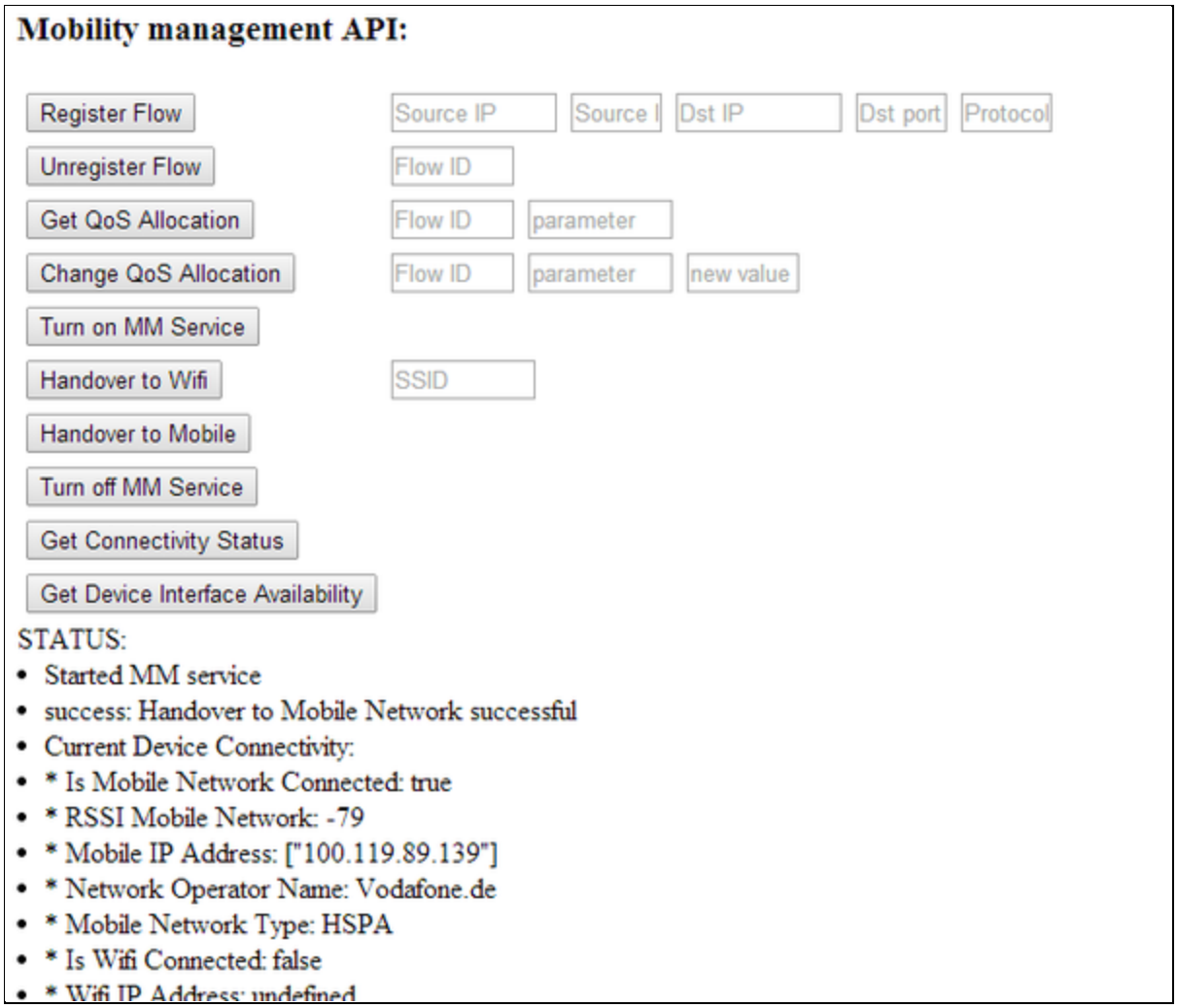
```
        function geolocationPositionCallBack(position) {
            $("#lat").html(position.coords.latitude);
            $("#long").html(position.coords.longitude);
            $("#alt").html(position.coords.altitude);
            $("#ll-accuracy").html(position.coords.accuracy);
            $("#alt-accuracy").html(position.coords.altitudeAccuracy);
            $("#heading").html(position.coords.heading);
            $("#speed").html(position.coords.speed);
```

```
            var d = new Date(position.timestamp);
            var dateString = d.toLocaleString();
            $("#timestamp").html(dateString);

            var pos = new google.maps.LatLng(position.coords.latitude,
position.coords.longitude);
            var marker = new google.maps.Marker({
                position: pos,
                map: map,
                title: 'Hello World!'
            });

            map.setCenter(pos);
        };
```

After updating the fields the function then updates the Google map item, by supplying it with the devices current latitude and longitude and updated the marker position.

For more information on the GeoLocation API please see the CDI Sensor Open Specifications.

2.3.4.3 *Basic Web Application: Mobility Manager*




```
<script type="text/javascript"

	src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.
js"></script>

<!-- <script type="text/javascript"
src="localhost:8080/webinos.js"></script>  -->

<script type="text/javascript" src="/static/webinos.js"></script>

<script type="text/javascript"
src="http://localhost:8080/webinos.js"></script>

<script type="text/javascript" src="cdi.js"></script>

<script type="text/javascript" charset="utf-8">

	$cdi.init();

</script>
```

```
<script>
$(document).ready(
	function() {
		function log_result(msg) {
			console.log(msg);
			$('#message').append('<li>' + msg + '</li>');
		}

		$('#registerFlow').bind('click',
			function() {
				$cdi.mm.registerFlow(
					$('#source_ip').val(),
					$('#source_port').val(),
					$('#dst_ip').val(),
					$('#dst_port').val(),
					$('#protocol').val(),
					function(result) {
						log_result("Registered flow
with id: "+ result);
					},
					function(error) {
						log_result("ERROR: " +
error.data);
					}
				);
			}
		);

		$('#unregisterFlow').bind('click',
```

```
                        function() {
                                $cdi.mm.unregisterFlow(
                                        $('#unregister_sessionid').val(),
                                        log_result("Success"),
                                        function(error) {
                                                log_result("ERROR: " +
error.data);
                                        }
                                );
                        }
                );

                $('#getQosAllocation').bind('click',
                        function() {
                                $cdi.mm.getQosStatus(
                                        $('#getQos_sessionid').val(),
                                        $('#getQos_parameter').val(),
                                        function(result) {
                                                log_result(result.flowId + ", "
+ result.parameterName+ ", " + result.paramterValue);
                                        },
                                        function(error) {
                                                log_result("ERROR:" +
error.data);
                                        }
                                );
                        }
                );

                $('#changeQosAllocation').bind('click',
```

```
                        function() {
                                $cdi.mm.modifyQosAllocation(
                                        $('#changeQos_sessionid').val(),
                                        $('#changeQos_parameter').val(),
                                        $('#changeQos_value').val(),
                                        function(result) {
                                                log_result("Success");
                                        },
                                        function(error) {
                                                log_result("ERROR:" +
error.data);
                                        }
                                );
                        }
                );

                $('#handoverWifi').bind('click',
                        function() {
                                $cdi.mm.handoverToWifi(
                                        $('#wifi_ssid').val(),
                                        function(result) {
                                                log_result(result);
                                        },
                                        function(error) {
                                                log_result("ERROR:" +
error.data);
                                        }
                                );
                        }
```

```
		);


		$('#handoverTGPP').bind('click',
			function() {
				$cdi.mm.handoverToTGPP(
					function(result) {
						log_result("success: "+result);
					},
					function(error) {
						log_result("Error handing over
to TGPP: "+ error.data);
					}
				);
			}
		);


		$('#startMmService').bind('click',
			function() {
				$cdi.mm.startMmService(
					function(result) {
						log_result("Started MM
service");
					},
					function(error) {
						log_result("Could not start MM
Service: "+ error.data);
					}
				);
			}
		);
```

```
		$('#stopMmService').bind('click',
			function() {
				$cdi.mm.stopMmService(
					function(result) {
						log_result("Stopped MM
service");
					},
					function(error) {
						log_result("Could not stop MM
Service: "+ error.data);
					}
				);
			}
		);


		$('#getDeviceConnectivity').bind('click',
			function() {
				$cdi.mm.getDeviceConnectivity(
					function(result) {
						log_result("Current Device
Connectivity: ");
						log_result(" * Is Mobile
Network Connected: "+result.isMobileNetworkConnected);
						log_result(" * RSSI Mobile
Network: "+result.rssiMobileNetwork);
						log_result(" * Mobile IP
Address: "+result.mobileIpAddress);
						log_result(" * Network Operator
Name: "+result.networkOperatorName);
						log_result(" * Mobile Network
Type: "+result.mobileNetworkType);
```

```
                                        log_result(" * Is Wifi
Connected: "+result.isWifiConnected);

                                        log_result(" * Wifi IP Address:
"+result.wifiIpAddress);

                                        log_result(" * WiFi SSID:
"+result.wifiSSID);

                                        log_result(" * WiFi Link Speed:
"+result.wifiLinkSpeed);

                                        log_result(" * RSSI WiFi:
"+result.rssiWifi);

                                },

                                function(error) {

                                        log_result("MM: Could not get
connectivity parameters: "+ error.data);

                                }

                        );

                }

        );


        $('#getInterfaceAvailability').bind('click',

                function() {

                        $cdi.mm.getInterfaceAvailability(

                                function(result) {

                                        log_result("Current Interface
Availability: ");

                                        log_result(" * Has WiFi:
"+result.hasWifi);

                                        log_result(" * Has Bluetooth:
"+result.hasBluetooth);

                                        log_result(" * Has Bluetooth
Low Energy profile: "+result.hasBluetoothLE);

                                        log_result(" * Has Mobile Data:
"+result.hasMobileData);
```

```
                                        },
                                        function(error) {
                                                log_result("MM: Could not get
connectivity parameters: "+ error.data);
                                        }
                                );
                        }
                );
        }
);


</script>
<script type="text/javascript"
        src="http://localhost:8080/webinos-api-
mm/test/web_root/cdi.js"></script>
</head>


<body>
        <section id="main" class="column">
                <article class="module width_full">
                        <header>
                                <h3>Mobility management API:</h3>
                        </header>
                        <div class="module_content">
                                <table>
                                        <tr>
                                                <td><button id="registerFlow"
class="button">Register Flow</button></td>
                                                <td><input id="source_ip"
placeholder="Source IP" size="10" />
```

```
                                                <input id="source_port"
placeholder="Source Port" size="3" />

                                                <input id="dst_ip"
placeholder="Dst IP" size="10" />

                                                <input id="dst_port"
placeholder="Dst port" size="3" />

                                                <input id="protocol"
placeholder="Protocol" size="3" />

                                        </td>

                                    </tr>

                                    <tr>

                                        <td><button id="unregisterFlow"
class="button">Unregister Flow</button></td>

                                        <td><input
id="unregister_sessionid" placeholder="Flow ID" size="6" /></td>

                                    </tr>

                                    <tr>

                                        <td><button
id="getQosAllocation" class="button">Get QoS Allocation</button></td>

                                        <td><input
id="getQos_sessionid" placeholder="Flow ID" size="6" />

                                            <input
id="getQos_parameter" placeholder="parameter" size="8" />

                                        </td>

                                    </tr>



                                    <tr>

                                        <td><button
id="changeQosAllocation" class="button">Change QoS
Allocation</button></td>

                                        <td><input
id="changeQos_sessionid" placeholder="Flow ID" size="6" />

                                            <input
id="changeQos_parameter"    placeholder="parameter" size="8" />
```

```
                                        <input id="changeQos_value"    placeh

                                    </td>

                                </tr>



                                <tr>

                                    <td><button id="startMmService"
class="button">Turn on MM Service</button></td>

                                </tr>



                                <tr>

                                    <td><button id="handoverWifi"
class="button">Handover to Wifi</button></td>

                                    <td><input id="wifi_ssid"
placeholder="SSID" size="8" /></td>

                                </tr>



                                <tr>

                                    <td><button id="handoverTGPP"
class="button">Handover to Mobile</button></td>

                                </tr>



                                <tr>

                                    <td><button id="stopMmService"
class="button">Turn

                                                        off MM
Service</button></td>

                                </tr>

                                <tr>

                                    <td><button
id="getDeviceConnectivity" class="button">Get Connectivity
Status</button></td>

                                </tr>
```

```
                                    <tr>

                                                <td><button
id="getInterfaceAvailability" class="button">Get Device Interface
Availability</button></td>

                                    </tr>

                               </table>

                         </div>

               </article>

               <div class="spacer"></div>

               <div id="status">

                      STATUS: <span id="message"></span>

               </div>

       </section>

</body>

</html>
```

2.3.4.4 *Basic Widget Instructions*

The previous sections have shown two examples to develop basic Web Applications using A-CDI GEi. These examples can be executed inside any HTML5-Compliant Internet Browser (for instance any recent release of Firefox, Opera, Chrome browsers). The A-CDI GEi also allows to execute the code as a Widget, using an embedded Widget Runtime Engine contained in the software distribution. To execute the previous examples as a widget, a new file must be created, called `config.xml`, containing the following instructions:

```
<widget xmlns="http://www.w3.org/ns/widgets" id="http://fi-
ware.eu/projects/i2nd/cdi/testing/myexample">

  <name>CDI Example</name>

   <description>CDI Usage Example</description>

   <author email="myname@somewhere.com">Myfirstname Mylastname</author>

   <content src="index.html"/>

   <icon src="cdifrontend.png"/>

</widget>
```

In the above code, the mandatory fields are the `id="..."` that must correspond to a unique id for every widgets (in the syntax of an URI, it does not need to correspond to a real URL), and the `content src="..."` that defines the HTML page that must be opened (typically index.html). All other fields are used to add descriptive information and should be filled with proper information; the `icon src="..."` can be optionally used to provide a graphical icon to the widget.

The HTML and Javascript code for a Widget is the same of a code for a Webpage, with one unique difference: as explained in the Connected Device Interfacing - Installation and Administration Guide, the inclusion of the definitions of the CDI-Webinos APIs must be done with a slightly different instruction (`/static/webinos.js`). The best way to enable the usage of the code for both webpage and widgets is to include the following lines, respecting also their order:

```
<script type="text/javascript" src="/static/webinos.js"></script>

<script type="text/javascript"
src="http://localhost:8080/webinos.js"></script>
```

This is also explained in the comments of the examples provided in the previous sections.

The `config.xml` file, the HTML (with the above line) and all Javascript files should be compressed into a single ZIP-format archive file. It is possible to use any name for this file. The final step to build the widget is to rename the ZIP-format file, changing its extention from '.zip' to '.wgt' (for instance, myexample.wgt). The widget has now been built and can be put everywhere in the memory of the device.

The widget will be searched by selecting the menu `Scan SDCard` after running the CDI-Webinos application on any Android device. A popup window will appear showing all available widget found; to install them, click on the checkboxes at the left side of their name. After this operation, the widget will appear inside the main screen of the CDI-webinos application and can be launched by clicking on its name.

3 Cloud Edge - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

3.1 Introduction

This page relates to the Cloud Edge GE which is part of the Interface to the Network and Devices chapter. Please find more information about this Generic Enabler in the Cloud Edge Open Specification.

3.1.1 Goal of this document

3.1.1.1 *What will be explained*

The goal of this document is to give enough information for programmers to develop and test Virtual Machines on Cloud Proxies HW boxes.

3.1.1.2 *What will NOT be explained*

This document expects programmers to fully understand Linux development as well as application visualization techniques (LxC).

It is also expected that a full cross development environment is installed by the programmers on a dedicated PC.

No User's Guide is provided, as the use of the Cloud Proxy by end users is expected to be described by the developers of applications and services running on it.

3.1.1.3 *Examples*

The manual shows how to create new VMs that tun on the Cloud Proxy, as described in the Programmers section; an example of how a VM can be installed and used can be fund respectively in the OSGi Installation and Administration and in the OSGi user and programmers manual, which can be considered a working live example for the programmers.

The distribution files for the Cloud Edge include the commented and documented source code for various examples:

- a simple ssh server giving a remote access to the shell of a virtual machine
- a simple NAS (samba, ftp) server
- a dedicated multimedia server

It is recommended that users are first trying to have an example running on their Cloud Proxy before trying to develop new ones.

These examples are NOT part of the official distribution and are here only to give the user a first working template he/she can start from when developing his/her own applications. These examples are therefore provided "as is" and are not intended to be used without modification in real products.

3.2 User Guide

As mentioned in the Introduction, a User's Guide is not provided, as the use of the Cloud Proxy by end users is expected to be described by the developers of applications and services running on it.

3.3 Programmers Guide

3.3.1 Create a CloudEdge Virtual Machine image for STB (HTTP web server)

A CloudEdge Virtual Machine image is composed of :

- a manifest file
- a rootfs

For instance, a CloudEdge VM image that provides a HTTP web server can be created :

a) extract the common rootfs template (rootfs-container) used by VMs :

```
$ mkdir rootfs-container

$ tar xzf rootfs-container.tar.gz -C rootfs-container
```

b) build the lighttpd open source webserver using the toolchain (cross-compilation) :

```
$ tar xjf lighttpd-1.4.28.tar.bz2

$ cd lighttpd-1.4.28

$ ./configure --build=i686-pc-linux-gnu --host=i686-cm-linux --without-
pcre --without-zlib --without-bzip2 --prefix=/usr --
libdir=/usr/lib/lighttpd

$ make
```

c) integrate lighttpd in rootfs-container :

```
$ install -d ../rootfs-container/{usr/sbin,usr/lib/lighttpd,var/www}

$ install
src/.libs/{mod_dirlisting.so,mod_indexfile.so,mod_staticfile.so}
../rootfs-container/usr/lib/lighttpd

$ install src/lighttpd ../rootfs-container/usr/sbin/lighttpd
```

d) create lighttpd.conf configuration file in rootfs-container/etc :

```
server.document-root = "/var/www"

index-file.names = ( "index.html" )

mimetype.assign = ( ".html" => "text/html" )
```

and add execute permission on it :

```
$ chmod +x rootfs-container/etc/lighttpd.conf
```

e) create S10lighttpd init script in rootfs-container/etc/init.d (in order to automatically run the webserver when the VM starts)

```
#!/bin/sh lighttpd -f /etc/lighttpd.conf
```

f) create an index.html HTML test page in rootfs-container/var/www :

```
<HTML>

 <HEAD><TITLE>index.html</TITLE></HEAD>

  <BODY>

   <H 1> Lighttpd Sample Home Page </H1>

  </BODY>

</HTML>
```

g) package the Virtual Machine rootfs in FTP server directory located on PC :

```
$ cd rootfs-container

$ tar czf <FTP server directory location>/rootfs-lighttpd.tar.gz *
```

h) create an UUID (uuidgen command) and the manifest.xml file associated to the VM :

```
<?xml version="1.0" encoding="utf-8"?>

 <manifest

  uuid="UUID generated by uuidgen"

  label="lighttpd"

  description="web server"

  rootfs="rootfs-lighttpd.tar.gz">

 </manifest>
```

i) package manifest file in the FTP server directory located on PC :

```
$ tar cf <FTP server directory location>/cloudedge-lighttpd-vm.tar
manifest.xml
```

3.3.2 Run a CloudEdge Virtual Machine image on STB (HTTP web server)

a) Download the CloudEdge web server VM image on STB :

```
$ curl --user smithj:secret http://<STB IP address>:8080/cloudedge/images
-d '{"image" : {"imageRef" : "ftp://<PC IP address>/cloudedge-lighttpd-
vm.tar"}}'
```

b) Create a VM instance of this CloudEdge web server VM image

```
$ curl --user smithj:secret http://<STB IP
address>:8080/cloudedge/instances -d '{"instance" : {"imageRef" : 1}}'
```

c) Start the VM instance :

```
$ curl --user smithj:secret http://<STB IP
address>:8080/cloudedge/instances/1 -d '{"start" : null}'
```

On PC, test this STB web server by connecting to http://<STB IP address> url in a web browser

4 Cloud Edge OSGi - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

4.1 Introduction

This document provides information for programmers who want to develop and test the OSGI.Equinox.Framework feature on I2ND Cloud Edge GE of FI-WARE. A User manual is not provided as the only user operations expected are through the use of the Eclipse framework, please refer to the framework for details on its usage. Examples of working with the OSGi.Equinox.framework will be provided in this guide. In order to be able to perform the actions below explained, the programmer should be familiar with the descriptions of the Cloud Edge and its OSGi framework, in particular it is required that the steps described in the Cloud Edge - installation and administration guide and in the Cloud Edge OSGi - installation and administration guide have been executed.

4.2 User Guide

As abovementioned in the Introduction, a User manual is not provided as the only user operations expected are through the use of the Eclipse framework, please refer to the framework for details on its usage.

4.3 Programmers Guide

Stop&Start the OSGi.Equinox.framework

From the SSH console (see the Cloud Edge OSGi - Installation and Administration Guide) , the OSGI-Eclipse-Framework can also be stopped and started by entering the following commands (the first stops and the second starts the OSGi)

```
/command/svc –d  /service/osgi

/command/svc –u  /service/osgi
```

To check the OSGi status you can run a Browser specifying the URL ‘http:// <OSGI-Container_IP_Address>/’. If the OSGi is stopped the browser reply indicating that the Web server is not responding while if OSGi is running the Brouser reply with un ‘HTTP Error 404’. Upload a new bundle

The last option to remotely work with the OSGi framework is running a Telnet connection to the 1266 port of the ‘<OSGI-Container_IP_Address>’. The OSGi.Eclipse.framework for development and deploying purposes is activating a ‘Telnet’ deamon. To remotely connect with the OSGi framework simply run a Telnet Client specifying the ‘<OSGI-Container_IP_Address>’ and 1266 port. The ‘osgi>’ prompt is presented and the 'help’ command lists you with all the options of the OSGi console. For example the ‘ss’ command lists all the installed bundles with the relative status.

To upload e new bundle un FTP server should be activated on the same LAN of the I2ND.CE (IP 192.168.1.15). Moreover in the download root directory of the FTP server the it.telecomitallia.osgi.fiware.helloworld_1.0.0.201311051546.jar file should be available.

1) To upload the new bundle from the Telnet client :

```
osgi> install
ftp://192.168.1.15/it.telecomitallia.osgi.fiware.helloworld_1.0.0.201311051546.jar
```

An output like

```
'Bundle id is 22'
```

will be displayed

2) To check the status of the bundle from the Telnet client:

```
osgi> ss
```

The new bundle is listed but not in the 'running'state

```
Framework is launched.

id      State       Bundle

0       ACTIVE      org.eclipse.osgi_3.5.2.R35x_v20100126

1       ACTIVE      org.tigris.mtoolkit.iagent.rpc_3.0.0.20110411-0918

……..

21      ACTIVE      org.mortbay.jetty.util_6.1.23.v201004211559

22      INSTALLED
it.telecomitallia.osgi.fiware.helloworld_1.0.0.201311051546
```

3) To run the new bundle from the Telnet client:

```
osgi> start 22
```

4) To check the OSGi status you can run a Browser specifying the URL 'http:// <OSGI-Container_IP_Address>/index.html', the "Hello,World!" message will be displayed.

4.3.1 Check the OSGi properties

To check the OSGi properties from the Telnet console:

```
osgi> getprop
```

A lot of information will be displayed including:

```
osgi.configuration.area=file:/opt/ti/osgi/configuration/
osgi.framework.version=3.5.2.R35x_v20100126
osgi.install.area=file:/opt/ti/osgi/plugins/
osgi.instance.area=file:/opt/ti/osgi/data/
os.version=2.6.35.14
java.runtime.version=1.7.0_25-b15
osgi.startLevel=5
osgi.bundles.defaultStartLevel=4
osgi.bundlestore=/opt/ti/osgi/configuration/org.eclipse.osgi/bundles
```

5 NetIC GE - OFNIC - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

5.1 Introduction

OFNIC provides a programmable interface to retrieve information and control Openflow Networks. The interface is based on the RESTful paradigm and is powered by a Web Server developed in Python programming language. OFNIC relies on the Openflow protocol in order to abstract and virtualize forwarding capabilities of the Openflow Network, but also SNMP protocol to retrieve more accurate statistics. This GEi is an extension to the NOX Openflow controller. OFNIC is composed of many modules which communicate whith each other with events. It is designed with an event-driven paradigm also to manage information that comes from the network. The OFNIC RESTful interface is an instance of the NetIC API (NetIC Generic Enabler Open Specifications can be found here). Goal of this document is to provide a useful guide for developers who want to build new applications based on the NetIC API, and a guide for users that might utilize the provided GUI application. Specific sections are dedicated to developer tools intended to help them during development, test and deployment.

5.2 User Guide

The NetIC RESTful API reference provides a page which describes how to use this API in details. It includes a set of commands, which can be used to manipulate and instantiate network resources physical or virtual. Since NetIC RESTful API is based on the HTTP protocol there are a rich variety of tools, which can be used to access the OFNIC GEi interface. The sections of the Programmer Guide will go into more detail on how to use and develop against NetIC RESTful API.

5.2.1 OFNIC GUI

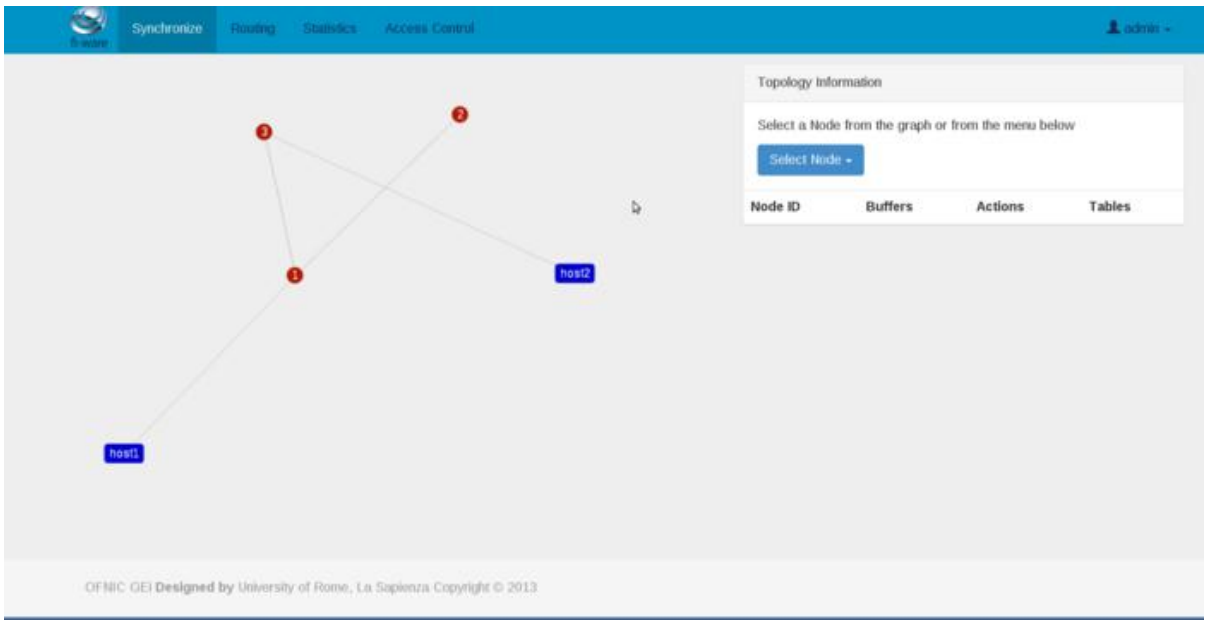
The OFNIC GEi is released with a GUI feature. The NetIC API users might utilize the GUI to navigate the exposed RESTful webservices of the GEi. The GUI has been developed in Javascript language, and uses the FI-WARE site template. The web GUI comes embedded in the OFNIC source package release, see the OFNIC Installation and Administration guide to download the package. The GUI relies on the OFNIC GEi, it should up and running correctly. To access the GUI from a web browser navigate to:

```
http://localhost/gui
```

The GUI page that appears is shown in the figure below.

The default username is 'admin' and can be authenticated with the password 'admin'. After logging in, one can note that the GUI is composed of four navigation tabs: *Synchronization*,*Statistics*, *Routing*, and *Access Control*. The Synchronization tab shows the network topology with an interactive diagram. Network nodes are clickable and additional information is shown if a node is selected. For example the number of Actions or Ports. Moreover the Displayed ports are clickable and show to which other network entity is the selected node connected to.

The Statistics navigation tab, concerns about the presentation of network statistics collected by the OFNIC GEi. The dynamic network topology is present so the user can select nodes he might want to monitor from the graph. The Statistics tab is shown in the figure below.



The presence of the Openflow network is not mandatory, if no network is attached to the OFNIC GEi, the following warning will appear on the GUI:

```
no network nodes detected
```

5.3 Programmer Guide

This section describes the way a programmer can interact ith the OFNIC GEi. It is assumed that the OFNIC GEi is up and running. See the OFNIC Installation and Administration guide for this.

5.3.1 Navigation with browser

The root path of the Web Server is:

```
https://127.0.0.1/netic.v1/doc
```

This page shows all commands that can be sent to the REST API of OFNIC. Note that not all the methods and commands might be supported directly from the browser, that is because some browser does not support natively the `DELETE` Http command.

Regarding `POST` command the parameters of the request might be:

- encoded in the url

```
field1=value1&field2=value2&field3=value3...
```

- formatted as JSON objects in the body of the Http Request.

```
{field1 : value1, field2 : value2, field3 : value3}
```

5.3.1.1 *Examples*

In this example the OFNIC GEi is connected to an Openflow network composed of three Openflow nodes. The nodes run Openvswitch for implementing the Openflow protocol and SNMPd and SNMP Sub-Agent to collect statistics. The OFNIC GEi is located at the same machine from which the browser is used. Using the browser, type in the location bar the following URL string and then press Enter.

```
https://127.0.0.1/netic.v1/OFNIC/synchronize/network
```

The result displayed, should be the list of nodes present in the network:

```
{
    "resultCode": 0,
    "displayError": "No error",
    "result": {
        "Paths": [],
```

```
        "Nodes": [
            1,
            2,
            3
        ]
    }
}
```

In order to retrieve information about the configuration of node with ID equal to 2, the following string should be typed in the URL container:

```
https://127.0.0.1/netic.v1/OFNIC/synchronize/network/node/2
```

The following JSON response should appear in the Web browser:

```
{
    "resultCode": 0,
    "displayError": "No error",
    "result": {
        "Port_Names": [
            "eth3",
            "eth2",
            "br0",
            "eth1"
        ],
        "Port_Index": [
            3,
            2,
            65534,
            1
        ],
        "Num_Buffers": 256,
        "Actions": 4095,
```

```
        "Num_Tables": 255
    }
}
```

The same operation can be repeated to get information about port 1 of node 2, with the following URL:

```
https://localhost:2222/netic.v1/OFNIC/synchronize/network/node/2/port/1
```

,the following result is expected:

```
{
    "resultCode": 0,
    "displayError": "No error",
    "result": {
        "Active": true,
        "Config": 0,
        "State": 0,
        "Speed": 0,
        "links": [
            0
        ]
    }
}
```

To retrieve statistics about a port of a certain node (for example port 1 of node 2) the following URL might be used:

```
https://localhost:2222/netic.v1/OFNIC/statistics/node/2/port/1
```

and the results that appears in the browser shows the transmitted and received bytes of the specified port:

```
{
    "resultCode": 0,
    "displayError": "No error",
    "result": {
        "Tx_bytes": 34,
```

```
        "Rx_bytes": 20
    }
}
```

As one can note from the examples, all response bodies are in JSON format and three fields are always present:

- resultCode: 0 for no errors, any other number for the occuring error.
- displayError: a string that displays the type of error
- result: the result of the request.

5.3.2 Apache HttpClient

Programmers who are interested in developing applications that rely on the NetIC RESTful API, might consider using well-known and already established REST clients and REST client libraries. Apache Software Foundation provides a powerful package HttpClient. HttpClient is an efficient, up-to-date, and feature-rich package implementing the client side of the most recent HTTP standards and recommendations. It is simple to use, so even not very experienced programmers can build simple applications in a few minutes. Some code snippets are reported below.

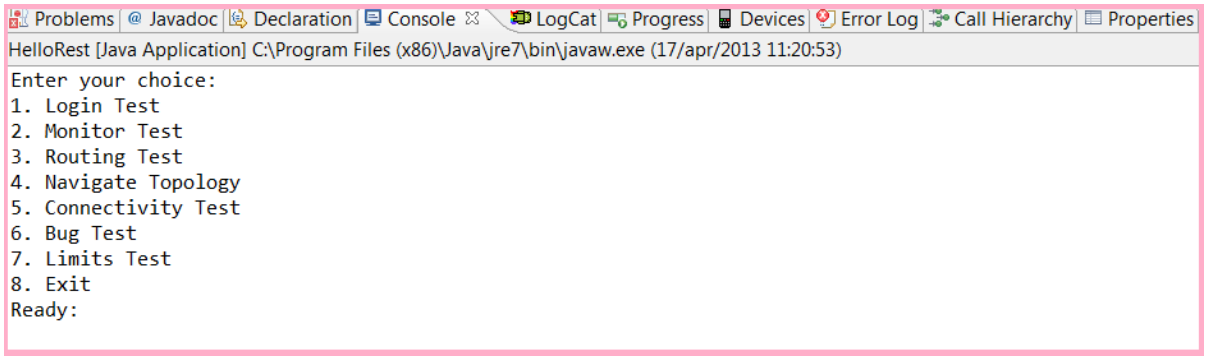
```
HttpClient client = new HttpClient();
HttpMethod request = new GetMethod("https://localhost/netic.v1/doc");
try {
    HttpResponse response = client.execute(request);
    HttpEntity entity = response.getEntity();
    if (entity != null) {
        BufferedReader rd = new BufferedReader
        (new InputStreamReader(response.getEntity().getContent()));
        String line = "";
        while ((line = rd.readLine()) != null) {
            textView.append(line);
        }
    }
} catch (ClientProtocolException e) {
    e.printStackTrace();
```

```
} catch (IOException e) {

    e.printStackTrace();

}
```

OFNIC GEi is released accompanied by a Java application based on Apache HttpClient, that was designed in the development and testing phase.

The source code of the application named NetIC_API_Tester can be downloaded from the FI-WARE repository (ofnic_netic_API_tester.zip file).

The application can be executed on Linux environment by running the script `test.sh` located inside the `src` folder. A list of test appears that can be executed by putting in the corresponding number.



The NetIc_API_Tester can be imported also in Eclipse[ECL] as a normal project. To do this, after the source code has been checked out locally, in Eclipse IDE go to File->Import... In the dialog box that appears under 'General' choose 'Existing Project into Workspace...' Following the wizard one should be able to import the project in the IDE and can manage it easily.

5.3.3 How to extend OFNIC GEi

OFNIC GEi is an extension to the NOX Openflow Controller, so the guide to extend NOX can be utilized to extend also OFNIC. The guide is located in this page:

```
https://github.com/noxrepo/nox-classic/wiki/Developing-in-NOX
```

6 NetIC GE - VNP - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

6.1.1 Introduction

The Virtual Network Provider, as an implementation of the Network Information and Control (NetIC) Generic Enabler (GE), will be hosted at the GEi owner premises.

The users of the NetIC API are network operators (Internet Service provider, Communication Service Providers) or in general service providers whose service requires connectivity with QoS guarantees. It is assumed that the users of the VNP GEi will use client programs that utilize the implemented NetIC API functions, the Programmer Guide below explains how to use the functions of the VNP to create their own service.

Before the VNP GEi software and especially the functions provided through the NetIC API are finalized, the potential users of the VNP GEi software will be provided direct access to the VNP functions. The User Guide below describes how to use this access and to have experience with the operation of the VNP.

The VNP will implement a subset of the NetIC API functions (for reference please check the NetIC architecture specification (see here), the NetIC Open Specification - see FIWARE.OpenSpecification.I2ND.NetIC, and the relevant parts of NetIC API Open specification (see here):

- query the *"available network resources"* (see NOTE)

  This is the "Synchronize" functionality of the NetIC API.

- serve on-demand and scheduled connectivity requests, and

  This is the "Create" and "Release" functionality of the NetIC API.

- change the configuration of virtual network resources (paths only).

  This is the "Provision" functionality of the NetIC API.

NOTE: in case of VNP the "available network resources" are usually virtual resources, as the VNP creates a network abstraction to allow customer-aware behaviour and flexibility in the network. It is in-line with the principles of the Software Defined Networking (SDN): the network is presented according to the customer needs and it may change on per customer basis. For some customers the virtual network is only a connection between two endpoints (and no internal routers/switches are presented), while others may be presented by nearly all details of the underlying physical network. The GEi offers networks based on Service Level Agreements (SLA), for the provided instance it is a fixed, static network representation.

As the VNP GEi software is available only remotely, the administration of users or subscribers will be done by GEi owner. NSN will give as much support as needed for these tasks. But it has the consequence that this User and Programmer Guide is not focused on these administrative steps, rather on the usage of NetIC API functions supported in the VNP GEi software.

6.1.2 User Guide

Note that as soon as the VNP GEi software is complete, it will provide a subset of the NetIC API and it is expected that only client programs will use it, so this section will be N/A, and only Programmer Guide will apply for the VNP GEi.

However, while the NetIC API support is not finalized a command line interface (CLI) is provided to access the Virtual Network Controller (VNC) and users may have experience with it. The following figure depicts this temporary setup.

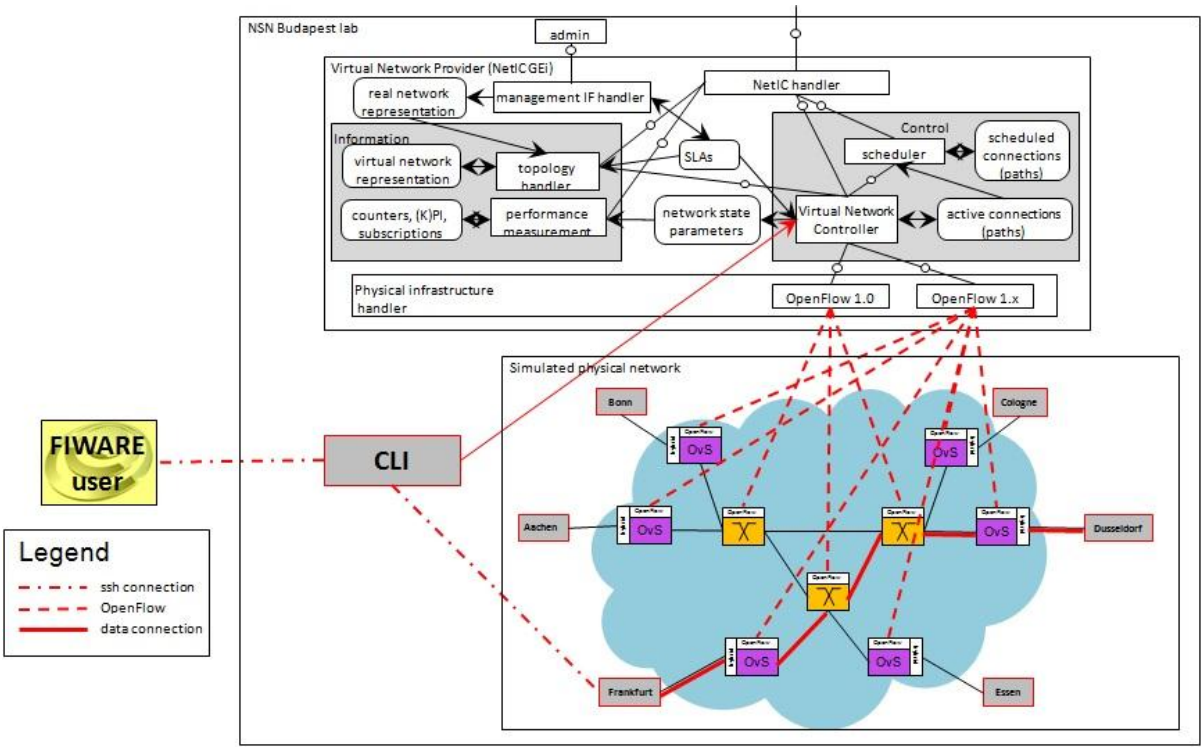


Figure 1: Direct access to VNC in Release 3.1

FIWARE users are provided with credentials to access the GEi owner laboratory, for details check the "Installation and Administration Guide" - available here (restricted access only for FI-PPP partners, needs login to FI-WARE forge) or in the "Instances" tab of the VNP GEi catalogue. The login process is also shown in the "Unit Testing Plan" - available here.

This access allows users to:

- submit commands to the Virtual Network Controller and
- connect to simulated customer nodes.

The VNC is shown on the figure as part of the VNP architecture, the description of the elements in the architecture are available in the NetIC Architecture description.

6.1.2.1 *Functions in the current release*

The users may add and remove paths in a virtual network, which is the representation of the simulated transport network (see the appropriate figures in the "Unit Testing Plan" - available here).

The customer nodes in the virtual network are:

- aachen.vnp.budapestlab.nsn.com,
- bonn.vnp.budapestlab.nsn.com,
- cologne.vnp.budapestlab.nsn.com,
- dusseldorf.vnp.budapestlab.nsn.com,
- essen.vnp.budapestlab.nsn.com, and
- frankfurt.vnp.budapestlab.nsn.com.

The virtual network provides a fully connected network (i.e. a full mesh topology) among these nodes, i.e. there is a direct virtual link between any pair of customer nodes. In release 3 the bandwidth and latency calculation is also implemented, for each virtual link a guaranteed 20Mb/s bandwidth can be reserved. The real latency in the lab is insignificant, for the Path Computation Engine it is defined as 10ms per link and 5 ms per node.

Users are allowed to add and remove paths among these nodes. The syntax of the available CLI commands are:

addpath <customernode1> <customernode2> [-b <GBRvalue>] [-l <msvalue>]

and

removepath <customernode1> <customernode2>

For the addpath command the both the bandwidth (-b) and the latency (-l) parameters are optional. It allows users to define the Guaranteed Bit Rate (GBR in Mb/s) and/or the maximum latency (in ms) for the requested path. If the bandwidth parameter is not defined, then the default 10Mb/s value is used. If the requested bandwidth for the path is higher then the available bandwidth for the specific virtual link, then the provided bandwidth is automatically reduced to the available bandwidth, or if no reserved bandwidth is available for the virtual link, then the request is ignored. The latency parameter must be at least 25ms, otherwise the request will fail (all paths include at least two links and a node in the physical network).

Note that these commands are the simplified versions of the "Create" and "Release" functionalities of the NetIC API, respectively.

Users may test the availability of paths by connecting to the customer nodes with an ssh connection first and issuing PING commands towards the appropriate customer nodes. To reach the customer nodes with an ssh connection, the management interface of that node must be used. Those interfaces are:

- aachen.management.budapestlab.nsn.com,
- bonn.management.budapestlab.nsn.com,
- cologne.management.budapestlab.nsn.com,
- dusseldorf.management.budapestlab.nsn.com,
- essen.management.budapestlab.nsn.com, and

- frankfurt.management.budapestlab.nsn.com

respectively.

For detailed description please see the "Unit Testing Plan" - available here.

In case an error is found please report it to the GEi owner contact person.

6.1.2.2 *Limitations in the current release*

The representation of the virtual network is fixed and there are no user specific representations. Parallel use of the system may reduce the available network, if necessary (based on request) we can reserve multiple separated virtual network, but with reduced bandwidth availabilities.

The complete network is emulated in the laboratory (i.e. there are no external customer nodes) and users cannot set up secure connections (e.g. VPN) yet.

The test environment is not dedicated to FI-WARE, but it is shared with other projects. The testing is "governed" by providing credentials to the laboratory with limited validity, but these limits are discussed and agreed with potential users in advance. This limitation is necessary as even though a virtual network is provided, the virtual network is the abstraction of a network that include physical switches (those 3 Pronto switches are shown in orange(may appear as yellow on your screen) and physical network connections, and as such those are limited resources.

6.1.3 Programmer Guide

The VNP GEi software does not support the NetIC API in the current release.

Please refer to the NetIC Generic Enabler Architecture and Open Specifications for the planned functions of the NetIC API.

In this chapter you will find guidance on how to program an own client which will contact the VNP GEi of the NetIC API. As all NetIC GE implementations, the VNP GEi will support only the subset of the NetIC API functions (an obvious example: as one of the goals of network virtualization is to hide the details of the underlying physical network from the virtual network users (i.e. Virtual Network Operators), the VNP GEi will not allow querying the status of the underlying network elements. The planned NetIC related functions and their applicability for VNP are listed below:

- Synchronization: in general the synchronization can be used to retrieve the available network resources and also information about their configuration. For VNP GEi the available network resources include the static, SLA based network representation and the QoS guaranteed connections. For these the synchronization function is unnecessary. If a VNP instance offers its services to anybody without previous SLA, then it would be necessary to create a default network representation and implement the synchronization function to provide that information.

- Create: It is a functionality for the creation of virtual network resources based upon existing physical or virtual resources. For VNP the virtual resource creation is a path reservation to create

QoS guaranteed connections on demand (note that QoS guarantee is valid only if it is available end-to-end).

- Destroy: It is for the destruction of virtual network resources. For the VNP GEi the virtual resource destroying is a path removal.
- Monitoring: it allows the retrieval of monitoring information collected from the network regarding failures and performance statistics. Currently VNP does not support monitoring functionality, one of the advantages of the network virtualization is the ability to hide network failures with automatic recovery.
- Provisioning: Ths functionality allows the configuration of network resources. For VNP the physical resources are the static virtual network representation that cannot be configured, while for the virtual resources the users may manipulate the QoS parameters of the reserved paths (with the limitations defined in SLA).
- Release: It allows the release of already configured network resource or resources, bringing them back to their default configuration. It is not applicable for VNP.

7 NetIC GE - VNEIC - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

7.1.1 Introduction

Virtual Network Element Information and Control (VNEIC) Generic Enabler instantiation (GEi) is capable to get configured to manage abstract logical network resources. A management request over a logical network resource from a FI-WARE system or operator may be terminated locally into the VNEIC GEi itself while in proxy operating mode or may start a full management session toward one or more ALU-1850TSS otherwise. The programming interface providing support for such kind of abstract logical network resources management is implementing the NetIC API handler accordingly to a RESTful paradigm by means of an HTTP server.

The VNEIC GEi doesn't include a User Interface, as all operations are performed at programmer's level, hence no User Guide is provided here.

In case further information is required, please refer to the contacts in the FI-WARE Catalogue page of this GEi.

Listed below more VNEIC references you may also like to follow

- NetIC Open Specification
- **VNEIC User and Programmers Guide**
- VNEIC Unit Testing Plan
- A Common Open Interface to Programmatically Control and Supervise Open Networks in the Future Internet
- 1850 Transport Service Switch
- Ruby on Rails Guide (v3.2.13)

7.1.1.1 *Bootstrap/Initialization*

The VNEIC at start of day doesn't hold nor provide any information being provided with an empty internal database.

As no auto discovery mechanisms are available, it is up to the FI-WARE user or system to properly bootstrap the VNEIC by providing information about physical resources a real network under control is made by. Such physical network resources definition allow the VNEIC to properly expose virtual network resources at NetIC API interface. Virtual resources are organized in a hierarchical form as described in NetIC Restful API Specification.

7.1.2 User Guide

There is not a User manual provided for the VNEIC GEi, as there are not direct interactions to be managed by the user (e.g. through a Graphical User Interface or a command line interface).

7.1.3 Programmer's Guide

7.1.3.1 *Network Resources*

A FI-WARE system or operator has the ability to fully manage virtual network resources by the NetIC API handler accordingly to a RESTful paradigm implemented by an HTTP server listening at port 3000 http://localhost:3000. Operations over a single virtual network resource include create, update, retrieve and delete.

(1)Network

Record

The Network resource record is defined accordingly to the following fields

- **id** Unique network identifier (format %d) automatically assigned by the system at creation
- **name** A string to name or label the network (format %s)
- **description** A text describing the network (format %s)

URL

The url to access the complete collection of currently defined virtual networks is

```
/networks
```

while a single item of the collection can be obtained by its identifier (.id field in the record) with

```
/networks/%d
```

Create

A sample session to create a new Network by providing its name and description is as follow

```
 $ curl -X POST -H "Accept: application/json" -d 'network[name]=FIWARE
Network #1' -d 'network[description]=This is a description about my
"FIWARE Network #1"' http://localhost:3000/networks

 {"created_at":"2013-09-11T10:21:54Z","description":"This is a
description about my \"FIWARE Network #1\"","id":1,"name":"FIWARE Network
#1","updated_at":"2013-09-11T10:21:54Z"}
```

please note as the .id field of the record definition is automatically assigned by the system.

Retrieve

Retrieval of an existing Network record by its identifier (.id field of the record definition)

```
 $ curl -X GET  -H "Accept: application/json"
http://localhost:3000/networks/1

 {"created_at":"2013-09-11T10:21:54Z","description":"This is a
description about my \"FIWARE Network #1\"","id":1,"name":"FIWARE Network
#1","updated_at":"2013-09-11T10:21:54Z"}
```

Destroy

An existing Network can be destroyed by its identifier (.id field of the record definition)

```
 $ curl -X DELETE http://localhost:3000/networks/1
```

Listing

Listing all currently defined networks

```
 $ curl -X GET  -H "Accept: application/json"
http://localhost:3000/networks

 [{"created_at":"2013-09-11T10:21:54Z","description":"This is a
description about my \"FIWARE Network #1\"","id":1,"name":"FIWARE Network
#1","updated_at":"2013-09-11T10:21:54Z"},

  {"created_at":"2013-09-11T10:34:41Z","description":"This is a
description about my \"FIWARE Network #2\"","id":2,"name":"FIWARE Network
#2","updated_at":"2013-09-11T10:34:41Z"}]
```

(2)Node

A Node is the entity representing a network element.

Record

The Node resource record is defined accordingly to the following fields

- **id** Unique node identifier (format %d) automatically assigned by the system at creation
- **name** A string to name or label the node (format %s)
- **description** A text describing the node (format %s)
- **ip** The IP address by which the Node can be accessed for management (format %d.%d.%d.%d)

URL

The url to access the complete collection of currently defined nodes of a given networks is

```
/networks/%d/nodes
```

while a single item of the collection can be obtained by its identifier (.id field in the record) with

```
/networks/%d/nodes/%d
```

Create

A sample session to create a new Node by providing its name, description and ip address is as follow

```
 $ curl -X POST -H "Accept: application/json" \

 >   -d 'node[name]=FIWARE Node #11'\

 >   -d 'node[description]=This is a description about my "FIWARE Node
#11"'\

 >   -d 'node[ip]=192.168.1.1'\

 >   http://localhost:3000/networks/1/nodes
```

please note as the .id field of the record definition is automatically assigned by the system.

Retrieve

Retrieval of an existing Node record of a given Network can be retrieved by its identifier (.id field of the record definition)

```
 $ curl -X GET  -H "Accept: application/json"
http://localhost:3000/networks/1/nodes/1

 {"created_at":"2013-09-12T09:47:23Z","description":"This is a
description about my \"FIWARE Node
#11\"","id":1,"ip":"192.168.1.1","name":"FIWARE Node
#11","network_id":1,"updated_at":"2013-09-12T09:47:23Z"}
```

Destroy

An existing Node of a given Network can be destroyed by its identifier (.id field of the record definition)

```
 $ curl -X DELETE http://localhost:3000/networks/1/nodes/1
```

Listing

Listing all currently defined nodes on a given network

```
 $ curl -X GET  -H "Accept: application/json"
http://localhost:3000/networks/1/nodes

 [{"created_at":"2013-09-12T10:16:13Z","description":"This is a
description about my \"FIWARE Node
#11\"","id":1,"ip":"192.168.1.1","name":"FIWARE Node
#11","network_id":1,"updated_at":"2013-09-12T10:16:13Z"},

  {"created_at":"2013-09-12T10:16:20Z","description":"This is a
description about my \"FIWARE Node
#12\"","id":2,"ip":"192.168.1.2","name":"FIWARE Node
#12","network_id":1,"updated_at":"2013-09-12T10:16:20Z"}]
```

(3)Equipment

An Equipment resource represent a component or a module of a network element; instances of an equipment could be line cards, etc.

Record

The Equipment resource record is defined accordingly to the following fields

- **id** Unique node identifier (format %d) automatically assigned by the system at creation
- **name** A string to name or label the node (format %s)
- **description** A text describing the node (format %s)

URL

The url to access the complete collection of currently defined equipments of a given network element in a network is

```
/networks/%d/nodes/%d/equipments
```

while a single item of the collection can be obtained by its identifier (.id field in the record) with

```
/networks/%d/nodes/%d/equipments/%d
```

Create

A sample session to create a new Equipment by providing its name and description is as follow

```
 $ curl -X POST -H "Accept: application/json" \

 >   -d 'equipment[name]=FIWARE Equipment'\

 >   -d 'equipment[description]=This is a description about my "FIWARE
Equipment"'\

 >   http://localhost:3000/networks/1/nodes/1/equipments
```

please note as the .id field in the record definition is automatically assigned by the system.

Retrieve

Retrieval of an existing Equipment record of a given Node in a Network is executed by its identifier (.id field in the record definition)

```
 $ curl -X GET  -H "Accept: application/json"
http://localhost:3000/networks/1/nodes/1/equipments/1
```

Destroy

An existing Node of a given Network can be destroyed by its identifier (.id field of the record definition)

```
 $ curl -X DELETE http://localhost:3000/networks/1/nodes/1/equipments/1
```

Listing

Listing all currently defined nodes on a given network

```
 $ curl -X GET  -H "Accept: application/json"
http://localhost:3000/networks/1/nodes/1/equipments
```

(4)Port

An Port resource represent an interface of an equipment in a node of a network.

Record

The Port resource record is defined accordingly to the following fields

- **id** Unique node identifier (format %d) automatically assigned by the system at creation
- **name** A string to name or label the port (format %s)
- **description** A text describing the port (format %s)

URL

The url to access the complete collection of currently defined ports of a given equipment in a node of a network is

```
/networks/%d/nodes/%d/equipments/%d/ports
```

while a single item of the collection can be obtained by its identifier (.id field in the record) with

```
/networks/%d/nodes/%d/equipments/%d/ports/%d
```

Create

A sample session to create a new Equipment by providing its name and description is as follow

```
 $ curl -X POST -H "Accept: application/json" \

 >   -d 'port[name]=FIWARE Port'\

 >   -d 'port[description]=This is a description about my "FIWARE Port"'\

 >   http://localhost:3000/networks/1/nodes/1/equipments/1/ports
```

please note as the .id field in the record definition is automatically assigned by the system.

Retrieve

Retrieval of an existing Port record of a given Equipment in a Node of a Network is executed by its identifier (.id field in the record definition)

```
 $ curl -X GET  -H "Accept: application/json"
http://localhost:3000/networks/1/nodes/1/equipments/1/ports/1
```

Destroy

An existing Node of a given Network can be destroyed by its identifier (.id field of the record definition)

```
 $ curl -X DELETE
http://localhost:3000/networks/1/nodes/1/equipments/1/ports/1
```

Listing

Listing all currently defined nodes on a given network

```
 $ curl -X GET  -H "Accept: application/json"
http://localhost:3000/networks/1/nodes/1/equipments/1/ports
```

8 NetIC GE - Altoclient - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

8.1.1 Introduction

This programming guide refers to the *Altoclient* GEi of I2ND-NetIC and is intended as an introduction for a programmer to understand the top level architecture and the basic programming conventions for using the *Altoclient* library. Therefore, the reader of this page should be familiar with the NetIC Library API Description. More detailed information about the software is available in the doxygen documentation at *./docs* of the Software Delivery File for program distribution (see next chapter).

8.1.2 User Guide

8.1.2.1 *Contents of the Software Delivery File*

Software delivery is done by using a compressed tar file named *libalto-x.x.x.tar.gz*. The x are denoting the version. The directory structure is as follows:

/ contains root data such as *Makefile.am* etc.

- *./include* contains header files for the application *alto.h* and *alto_util.h*
- *./src* contains all source files including internal header files
- *./contrib* contains test programs
- *./docs* contains documentation (doxygen)
- *./build* directory for all objects and the library libalto, this directory is generated after the make process has been run

The link for downloading the newest version of the Software Delivery File can be found in the download section of the *Altoclient* entry in the FI-WARE Catalogue.

8.1.2.2 *How to Start Using the Altoclient Library Functions*

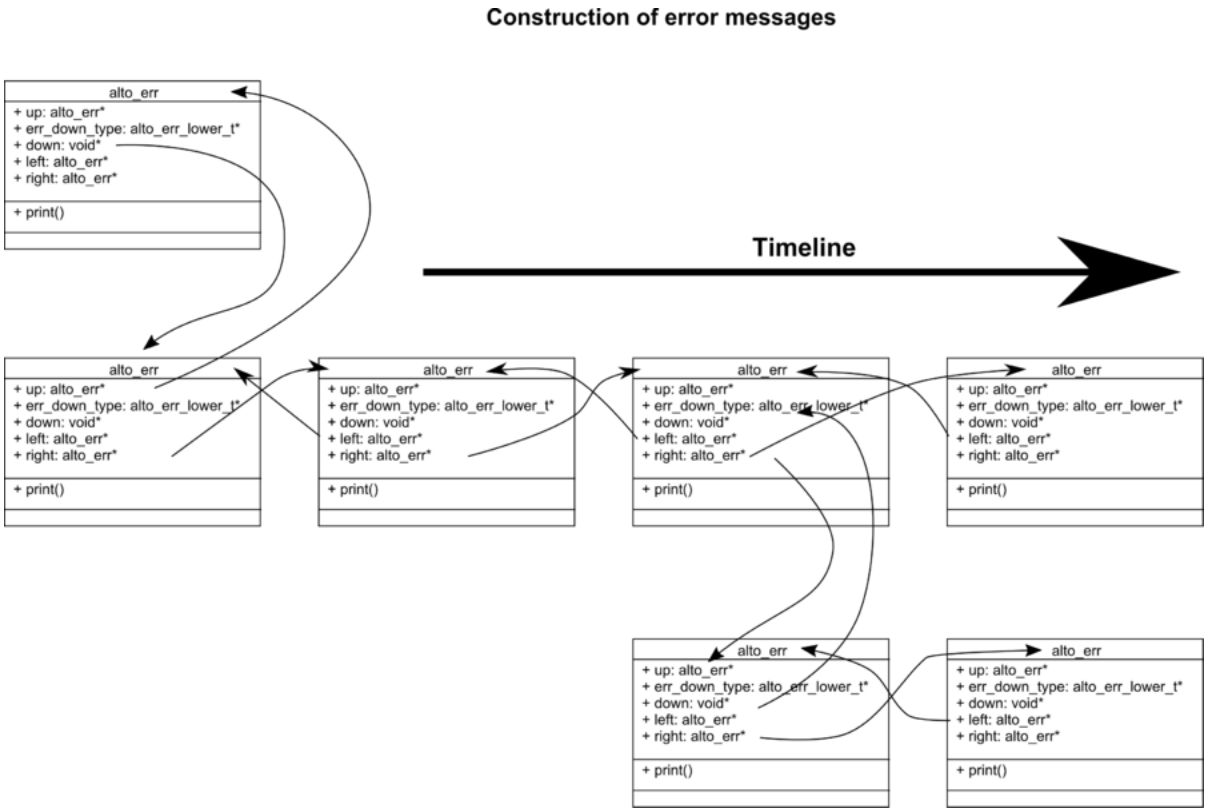
When starting to use functions from the Altoclient library, you should first get familiar with the library functions and the related parameters these functions require or provide. You can find this information in the NetIC Library API Description. Furthermore this Wiki page also contains information about the useful sequence of calling those functions in order to get useful results.

Furthermore, the Software Delivery File mentioned in the previous section contains in section *./contrib* the test program *alto_test_easy* which contains a well-documented example of using the function calls of the *Altoclient* library.

8.1.2.3 *Error Messages*

The reader should be familiar with the Error Reporting described in the NetIC Library API Description to understand the motivation for this error concept. Although the *alto_err objects* are given to the client application program the detailed description is given here rather in the API description as it is expected,

that normally the user only applies the print function to get details of the errors occurred but does not access the error objects by itself. Given is the following situation:



The *alto_err* object on the top is given via the API to the user, indicating that something has gone wrong in the API function (order 0). The down pointer leads to another *alto_err* object, which is created by a function called by the API function (order 1). Unfortunately, in this order-1-function not a single error occurred, but 4 in total. So the other error messages are chained in this level with the left and right pointers. The third error on this level points to an even lower level error. This scheme can be extended arbitrarily providing a strong tool to describe all possible errors. The down pointer is of type *void**, as it may point to errors generated by functions of the JANSSON and Curl library. In this case the *err_down_type* variable must be set to *JANSSONERR* or *CURLERR* respectively. If the *alto_err* points to a *alto_err* object *ALTOERR* must be specified. As can be seen from the image, it is absolutely required that the user calls the close function to delete all error messages. Only this method avoids any memory leaks.

8.1.3 Programmer's Guide

8.1.3.1 ***Basic Programming Conventions***

The programming language is C using the gcc compiler. Object oriented programming was applied, although C does not natively support this. The advantage of a well structured code is much more valuable. A typical definition of a class looks like this:

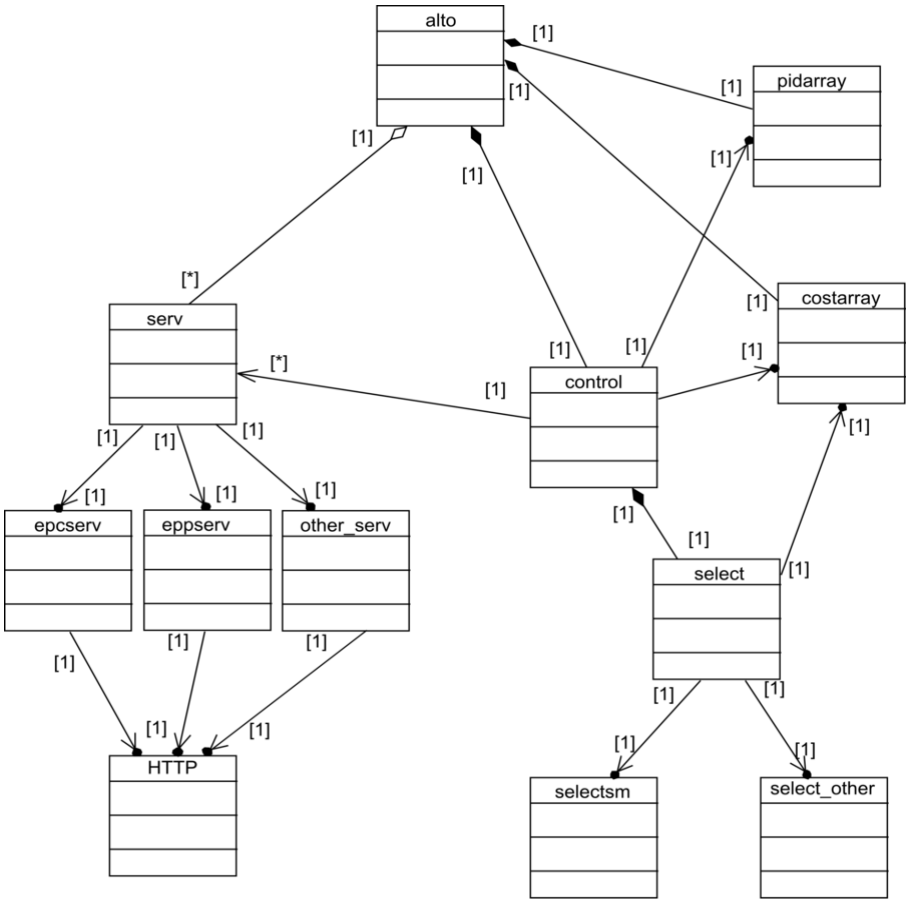
```
typedef struct alto_sample {

  int i;
```

```
  ...
}alto_sample;


void alto_sample_sample(alto_sample* self, ....); //constructor

int alto_samplefunc(alto_sample* self, <further parameters>); //member
function

void alto_close(alto_sample* self) //destructor
```

All functions and typedefs start with *alto_* in order to avoid any name conflicts with the application programmer's definitions. To ease reading in the following the prefix *alto_* is often omitted. All objects are typedef'ed to allow an easy instantiation by "*alto_sample MySample*". The constructor of the object is named by doubling the class name (without *alto_*). The destructor has the ending *_close*. If the object is created on the heap, the object needs to be freed thereafter. All member functions start with a pointer to the object.

8.1.3.2 *Class Diagram*

In the following the class diagram is explained.



On the top there is the *alto* class, which provides the API functions for the application programmer. On the left there is a general *serv* module, which is a front end for all ALTO request functions. There may be several *serv* modules, each module providing a special ALTO service on a special ALTO server. The *serv* class delegates the requests to special *serv* modules for each supported ALTO service. Currently an endpoint cost service class and an endpoint property service class is implemented. This delegation pattern was selected as the programming language C does not support object oriented programming natively and in this way a complex implementation of inheritance can be avoided.

All *serv* modules have a 1:1 association to the *HTTP* class, where the HTTP functionality is implemented. All *serv* modules must first be instantiated by the user and then be added to the *Altoclient*. This way it is possible to add and remove ALTO services at the user's intention.

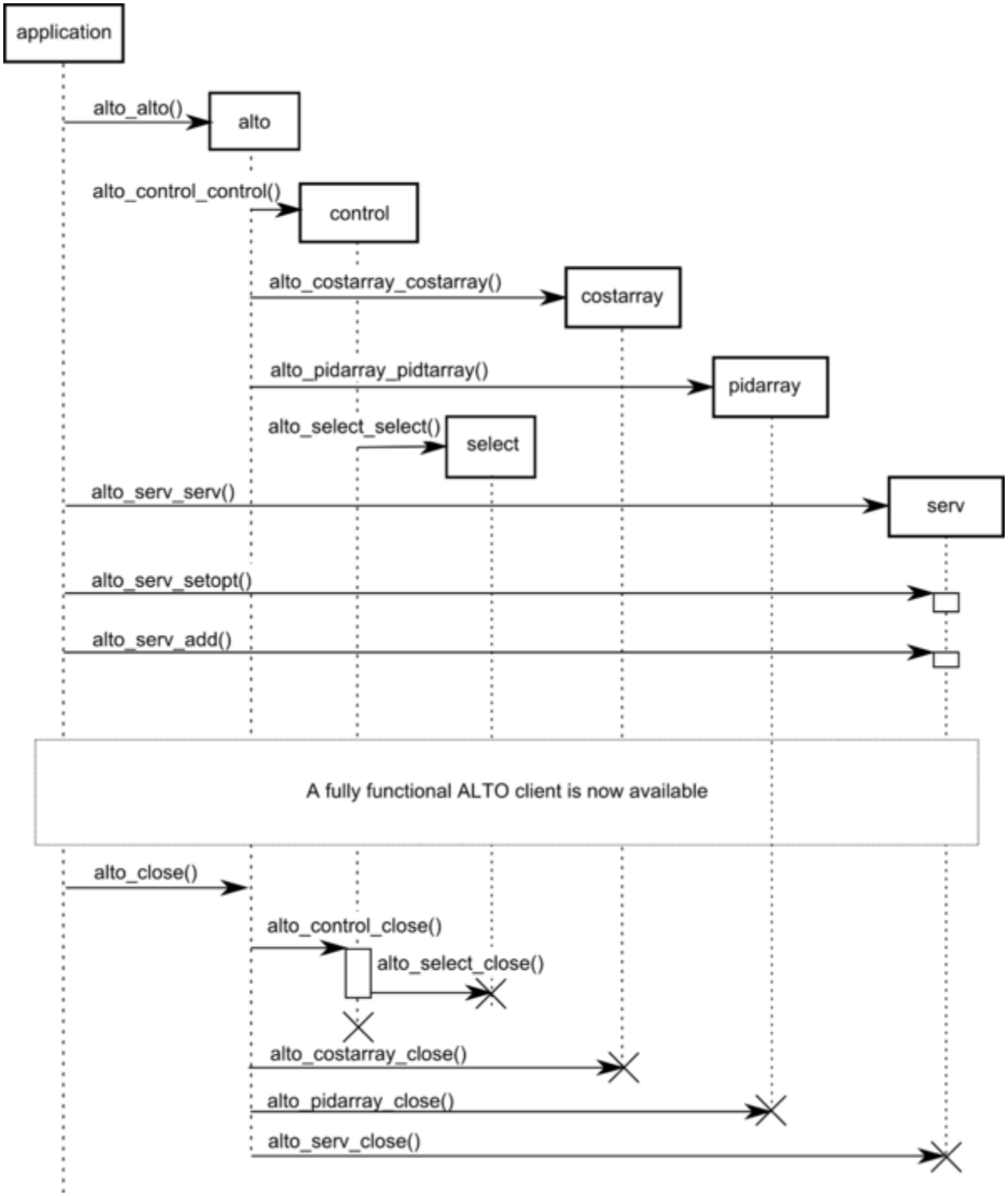
In the middle of the class diagram is the *control* module. It is connected to the *costarray* class, which holds all cost information retrieved by the different *serv* modules. Below is the generic *select* class which delegates its requests to different select modules. Currently only a simple select algorithm is available, which only searches for the node with the lowest cost, but does not take any other information into account.

The pidarray class holds the PID information generated by the endpoint property service. It is connected to the general alto class and to the control class.

8.1.3.3 ***Sequence Diagrams***

(1)ALTO Client Initialization and Destruction

The sequence diagram shown below describes, how an ALTO client object is generated and destructed. As already stated in the paragraph above, the modules *serv* and *select* are simply delegation modules which delegate their functions to the implementation modules *epcserv*, *eppserv* and *selectsm* (further modules to be added in the future). In this sequence diagram, this delegation pattern is not shown.

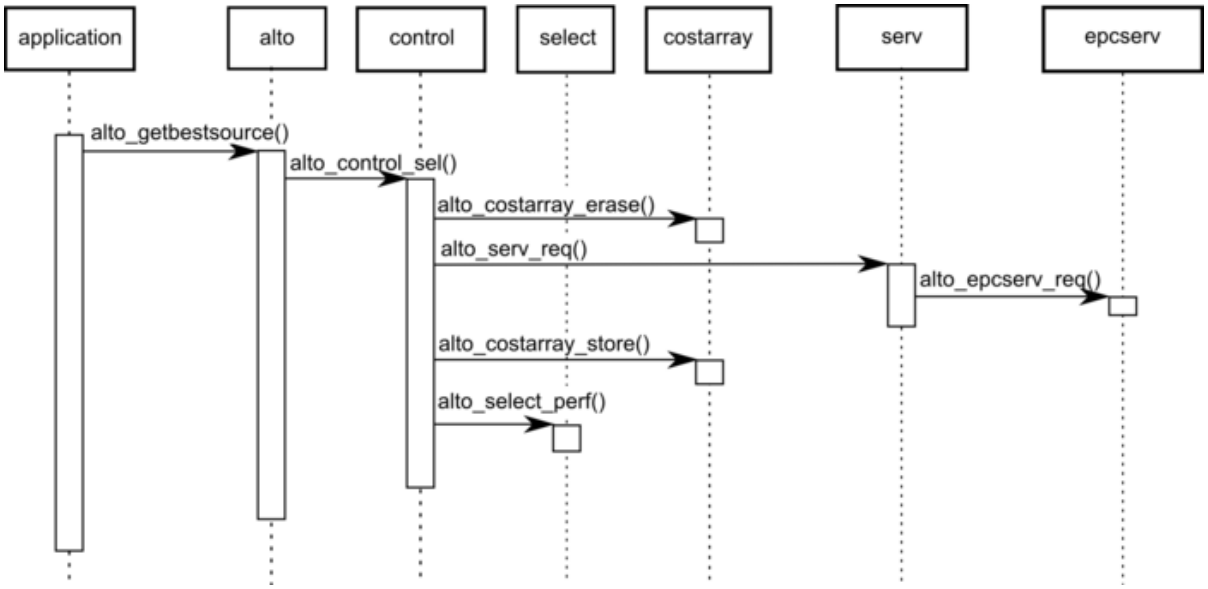


In the beginning, the application, which uses the ALTO client, calls the constructor *alto_alto()*, which in turn generates the *control*, *select*, *pidarray* and *costarry* modules. In addition to the sequence shown above, then also ALTO client options may be set using *alto_setopt()*. After the ALTO client has been established, the application calls the *alto_serv_serv()* constructor, which generates the *serv* modules for the alto client. Thereafter the options for the service have to be set, for example the URL of the resource directory of the ALTO server. To bind the *serv* module to the alto client, the function *alto_serv_add()* is called. The *serv* module sends a short test to the ALTO server (not shown in this sequence diagram) and upon success the *serv* module registers itself in the *control* and *alto* module. More than one service module can be created

and added to the ALTO client in the same manner. Thereafter a fully functional ALTO client is available for operation.

The destruction procedure starts with a function call *alto_close()* by the application program. This function destructs all modules and releases all system resources. The cascading calls of destructors show, that the user must call *alto_close()* and that simply freeing the memory associated with the *alto* structure is not sufficient.

(2)ALTO Endpoint Cost Service Request
In the following the sequence of an ALTO server call will be described to get the best source for the required information. Currently the functions *alto_getbestsource()* and *alto_getbestsink()* map to a single function named *alto_guidance()*, which is not shown here.



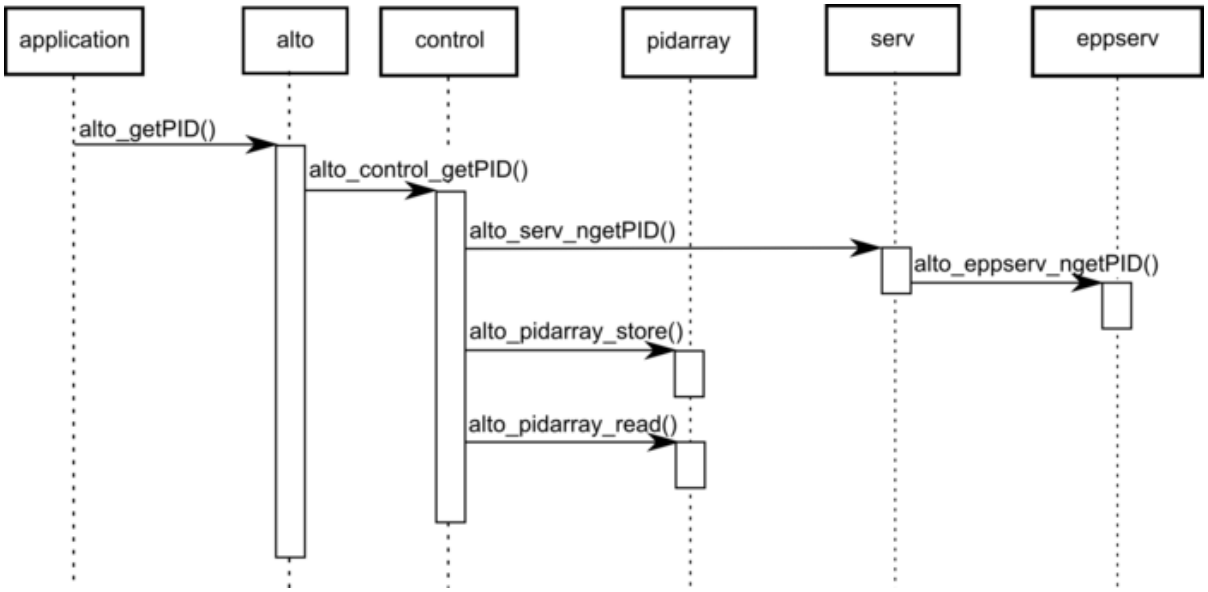
At first the application calls *alto_getbestsource()* of the alto object. The call is transferred to the *control* module which executes all required steps. First the cost data held in the *costarray* module are erased. Then the *serv* module is called with *alto_serv_req()*. This module delegates the call to the *epcserv module*, which implements the endpoint cost service. The result is given via the *serv* module to the *control* module, which uses *alto_costarray_store()* to save the results in the *costarry* module. There may be several *serv* and *epcserv* modules. Then all services are polled and their results are stored in the *costarry* module. After that, the *control* module initiates the select process by calling the *select* module using *alto_select_perf()*. The *select* module is also a delegating module which handles different select procedures realized by different modules. Currently, there is only a *selectsm* module, which realizes a simple minimum algorithm. For simplicity, this delegation is not shown in the diagram above. In the end the *control* module returns its results to the *alto* module and then to the calling application.

(3)ALTO Endpoint Property Service Request
The endpoint property service request is realized with two function calls, *alto_getPID()* and *alto_equalPID()*. The first function returns the PID (Provider-defined Identifier) of a network address and the second function tests if two nodes belong to the same PID. Both functions poll the information of all endpoint

property services, which are configured in the ALTO client. The return value of the *alto_getPID()* function is a linked list of results for every endpoint property service which was contacted for information.

```
typedef struct alto_PIDtable {

    alto_serv* service; //pointer to the service module

    char* PID; //resulting PID

    struct sockaddr addr; //requested address

    struct alto_PIDtable* next_entry; //pointer to the next element in
the linked list

} alto_PIDtable;
```

The following sequence diagram shows how the information is gathered:



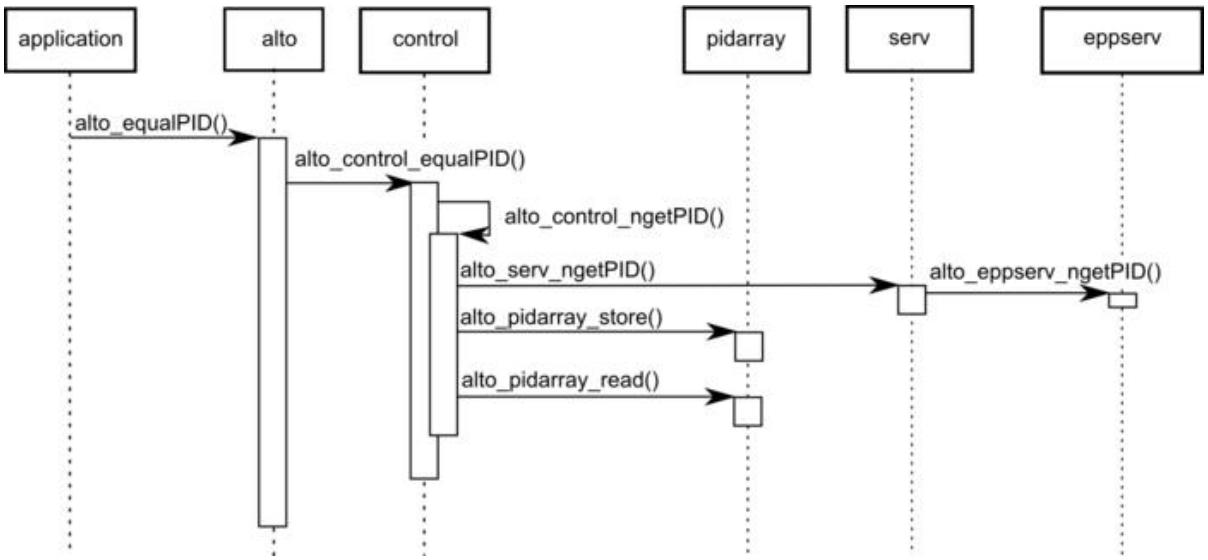
The application calls the *alto_getPID()* function which in turn calls the *alto_control_getPID()* function of the *control* module in the client. The control module calls the *serv* module and the *eppserv* module. The last module calls the ALTO server. Thereafter the *control* module stores the information in the *pidarry* module. Here the case is given, that only one endpoint property service module is registered. If there are more available further calls of the function *alto_serv_ngetPID()* and *alto_pidarray_store()* would be issued. In the last step the *pidarray* is read by the *control* module and the information is returned via the alto interface to the application.

The other function, which forms the endpoint property service request, is named *alto_equal_PID()*. It evaluates if two nodes belong to the same PID. Also this function returns a linked list:

```
typedef struct alto_PIDequal {
```

```
    alto_serv* service;//pointer to the service module

    int PIDequal; // =1 if the nodes belong to the same PID else =0

    struct alto_PIDequal* next_entry;//pointer to the next element in the
linked list

} alto_PIDequal;
```

The following figure shows how the information is gathered:



The application calls the interface function *alto_equalPID()*. Thereupon the *control* module invokes its function *alto_control_ngetPID()* which requests the PIDs for the two nodes from all initialized eppserv modules. Then, on a per *serv* module base it is investigated, which PIDs are equal. The results are given as a linked list to the application.

9 S3C GE - API Mediation - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

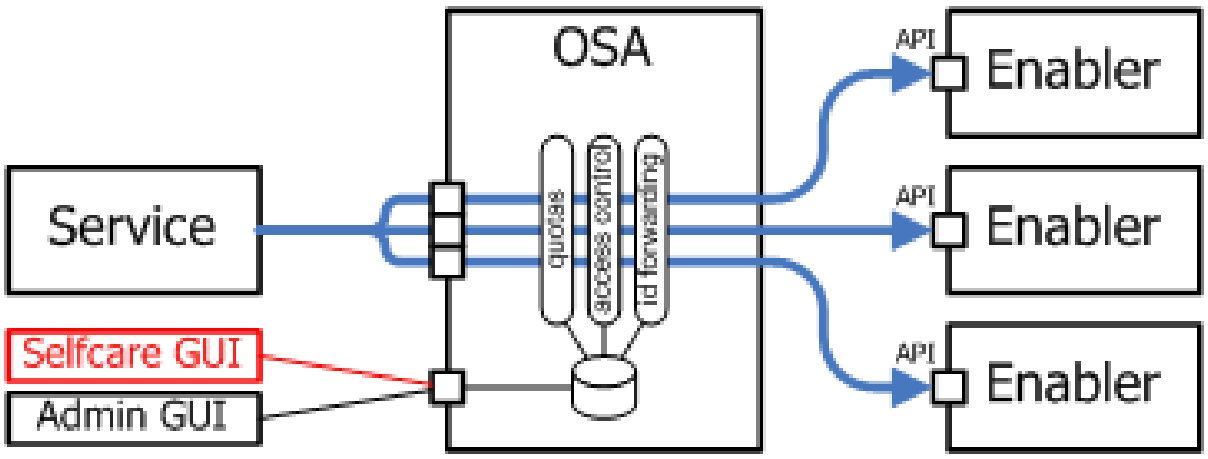
9.1 Introduction

9.1.1 What is Open Service Access?

Open Service Access (AKA OSA) is a Web Services gateway designed as a security element of your network, dedicated to WebService protection and publishing:

- Let web services developers be focused on the core business of their services and let OSA take care of security and protection of those services,
- Let web services user access to all your services form different network works in a single manner and doesn't bother providers with those issues

It comes with two parts: a mediation platform based on apache2, and a management application, as illustrated in the following picture:



9.2 User guide

9.2.1 What does Open Service Access?

OpensServicesAccess can be view as a web services dedicated proxy. It can:

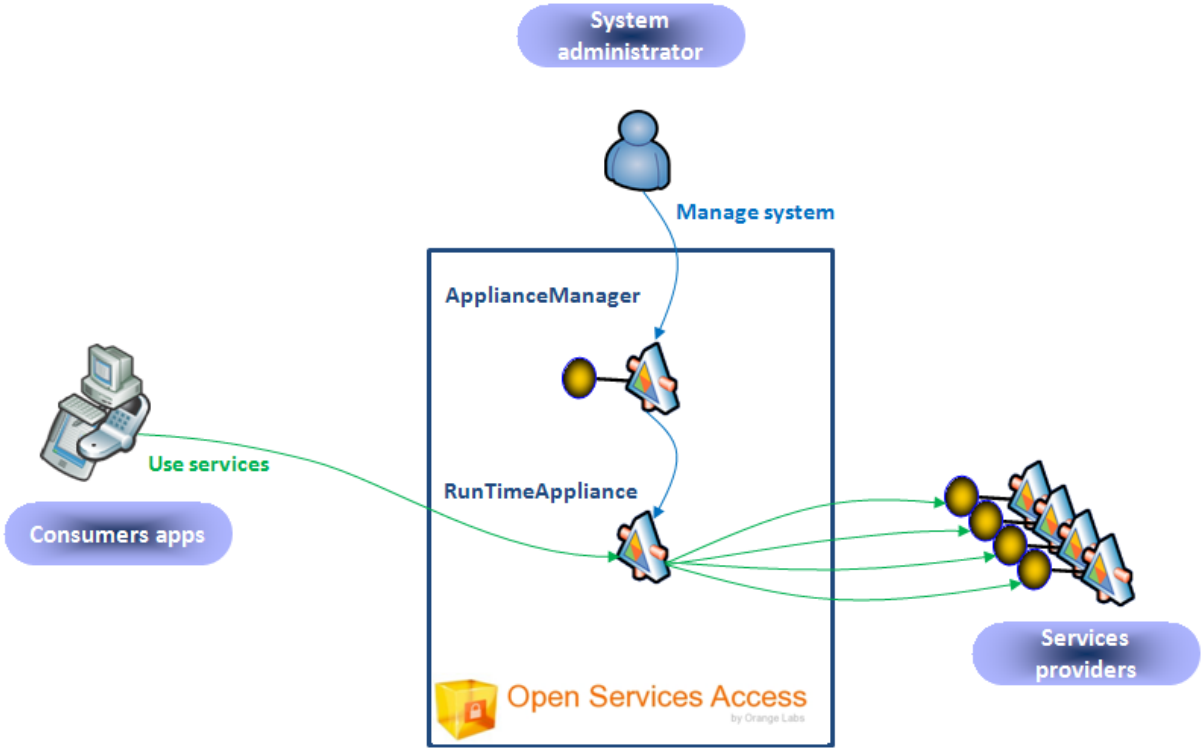
- Publish backends on different networks (default configuration allow up 2)
- Ensure encryption (https)
- Verify authentication with basic authentication
- Check authorizations

- Apply global quotas (per second, day and month) for a backend
- Apply user quotas (per second, day and month) for a backend
- Forward consumer identity to provider
- Forward publishing endpoint to provider
- Provide advance service usage logging to administrators.
- Offers simple GUI to adminstrators

9.2.2 How does Open Service Access works?

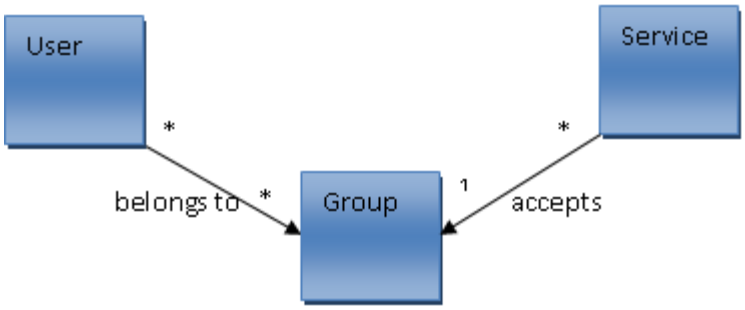
OSA works as a reverse proxy to and protect your resources, REST or SOAP. To do so, you have to provide information to expose the services, throught the AppliaceManagerAdmin. During the installation, be carefull to configure properly the admin password: it is needed to connect to the administration portal at the following url: https://*ServerName*/ApplianceManagerAdmin It is composed of 2 modules working together:

- RunTimeAppliance: which is in charge to manage access to web services at runtime. It is mainly based on Apache reverse proxy enhanced with new facilities,
- ApplianceManager: which is the administration application, including GUI and REST web services.



9.2.3 Authorization model

Each service deployed in the Open Service Access platform can be protected by an authorization. To do this, a user and authorization management has been implemented in the platform. If a service needs authorization, when using it, a prompt asks for a user. This user has to be a member of a group. A service is attached to a group.

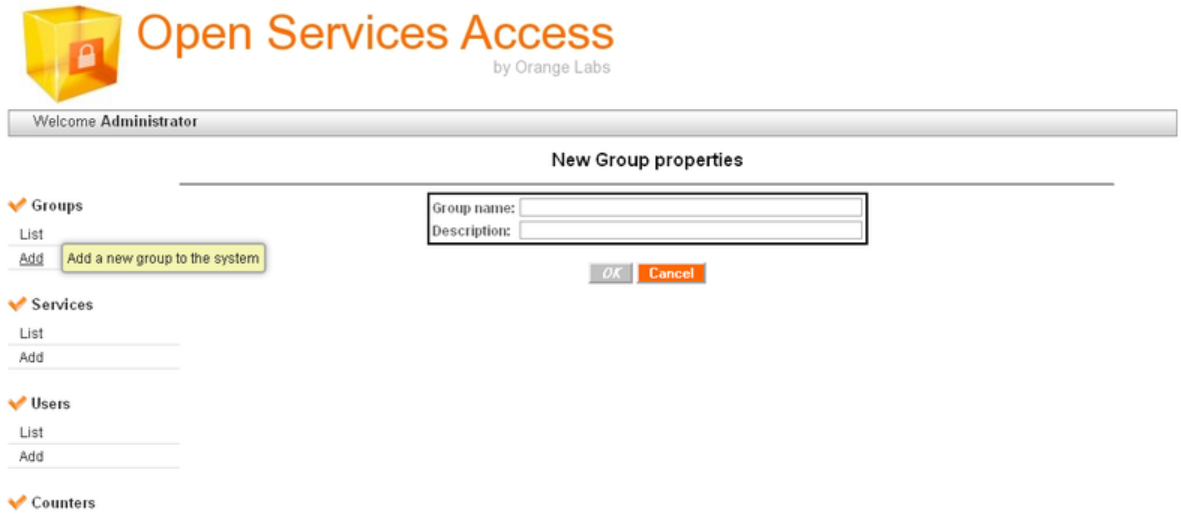


9.2.4 Submitting a service

This section gives you the different steps to submit a new service.

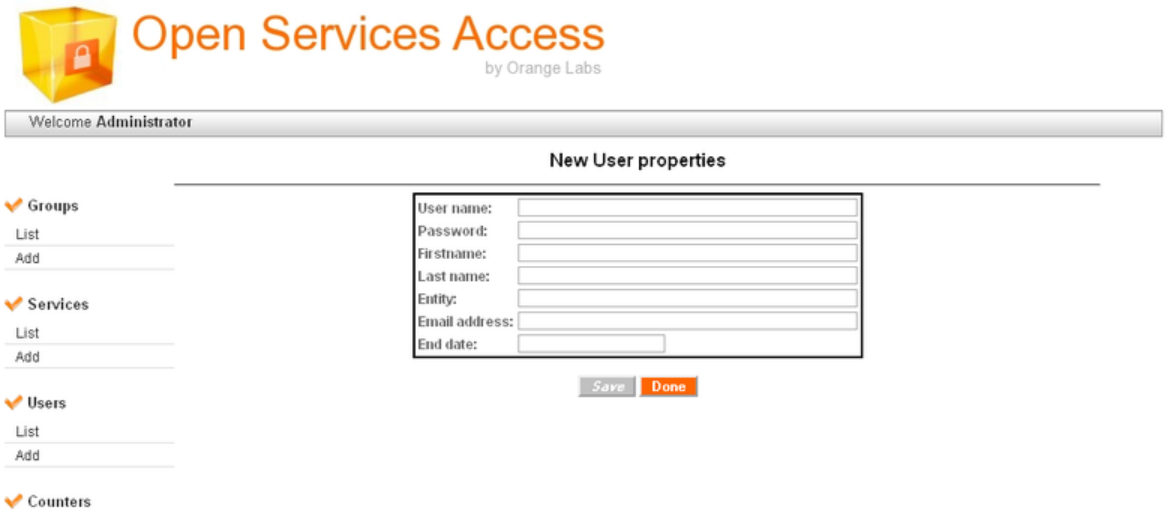
9.2.4.1 *1/ Create a new group*

First thing to do is to declare a new group. This group will be attached to the new service. You can of course use an existing group, but this would mean to share the user list with different services.

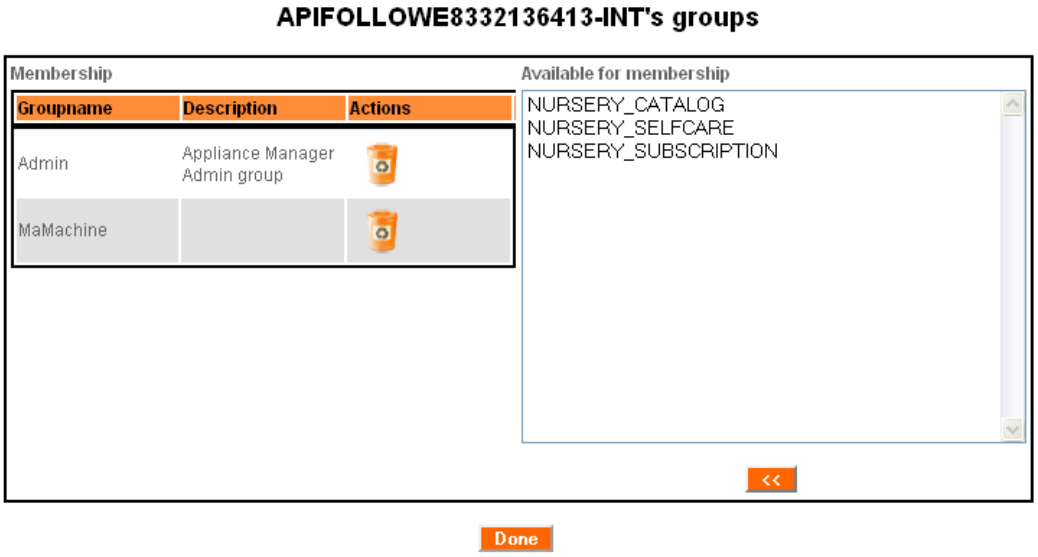


9.2.4.2 *2/ Fill the group with consumers*

If you want to allow a new consumer to use the service, you have to create a user (**Users** menu, **Add** section). Then, edit the user and add it to the group previously created.



In the following picture, we can see that the user called APIFOLLOWE8332136413-INT is member of the groups Admin and MaMachine. Therefore he will be able to consume all services attached to these groups.

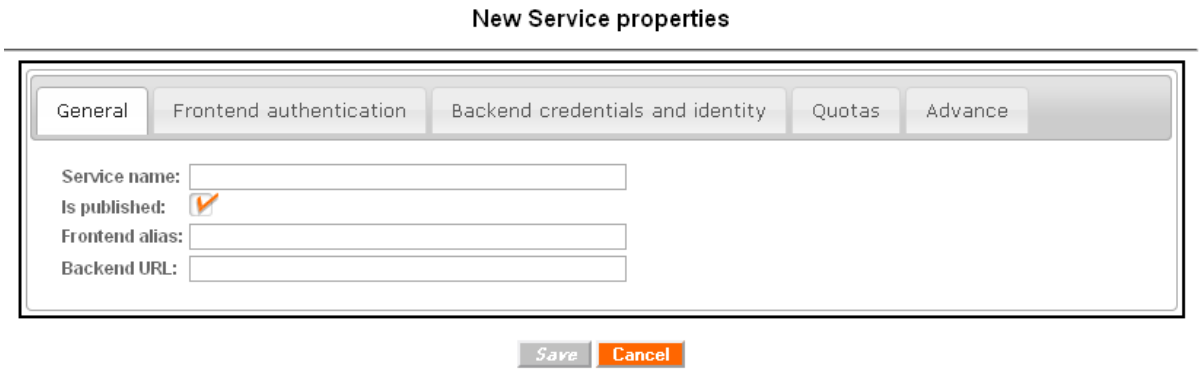


9.2.4.3 *3/ Publish a service*

The publication of a service is rather simple, and necessitates very few information, although advanced usage can be configured as well. As seen in the following picture, three parameters only are enough: a service name (the id), an alias, and of course the provider URL. By default, the checkbox *IsPublished* is checked, wich means that the service is usable. For example, if my OSA is installed on the server *ServerName* if I want to publish an SMS service which is located at the URL http://168.192.1.1/sms-oneapi (this is a local IP), I may enter this:

- Service name: SMS
- Frontend alias: SMS-V1
- Backend URL: http://168.192.1.1/sms-oneapi

Then save. At this moment, I can use the URL http://ServerName:81/SMS-V1 to consume the service.



9.2.4.4 *4/ Advanced parameters*

The previous steps give the minimum to be able to publish a service. The tabs on the picture give a view of the possibilities that are allowed to the administrator.

- Frontend Authentication enables you to select a group for protection of the service. If Enable user authentication is checked, the service consumer will have to use credentials with http basic authentication to consume the service.
- Backend credentials and identity can be necessary if the backend service needs authentication itself. Credentials can be provided at this place.
- Quotas: On this tab, the administrator can enter the maximum number of requests allowed to consume the service. The quotas can be global or per user. Three numbers are asked: usage by second, day and month.
- Last tab allows user to personalize apache directive to put more information in the http headers.

9.3 Programmer guide

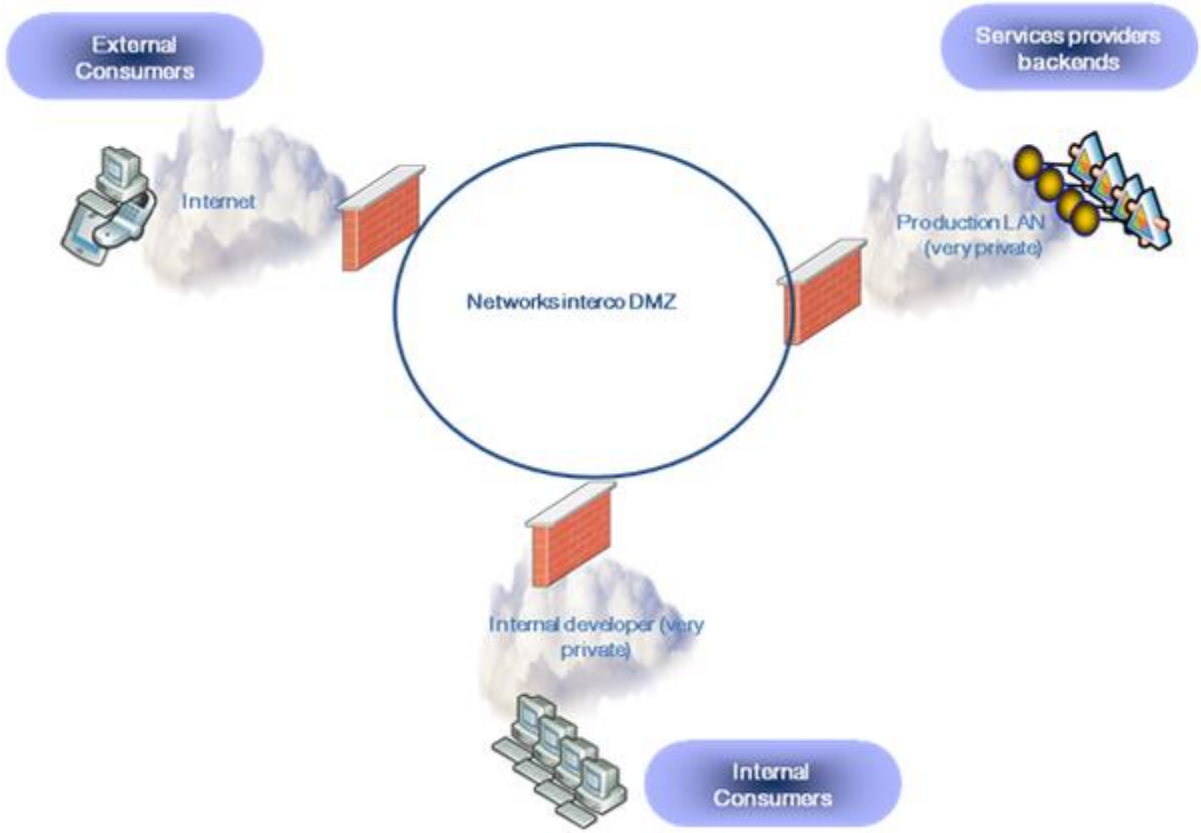
The API Mediation is a tool, but not an enabler in itself in the sense of that it is usable with an administration interface. Anyway, this section gives a more technical view of the possibilities of the tool.

9.3.1 Typical Use Case

Let's imagine that you have in your internal network some high value added services and you want to give access to all those services :

- Internally to your own developers,
- Externally to partners.

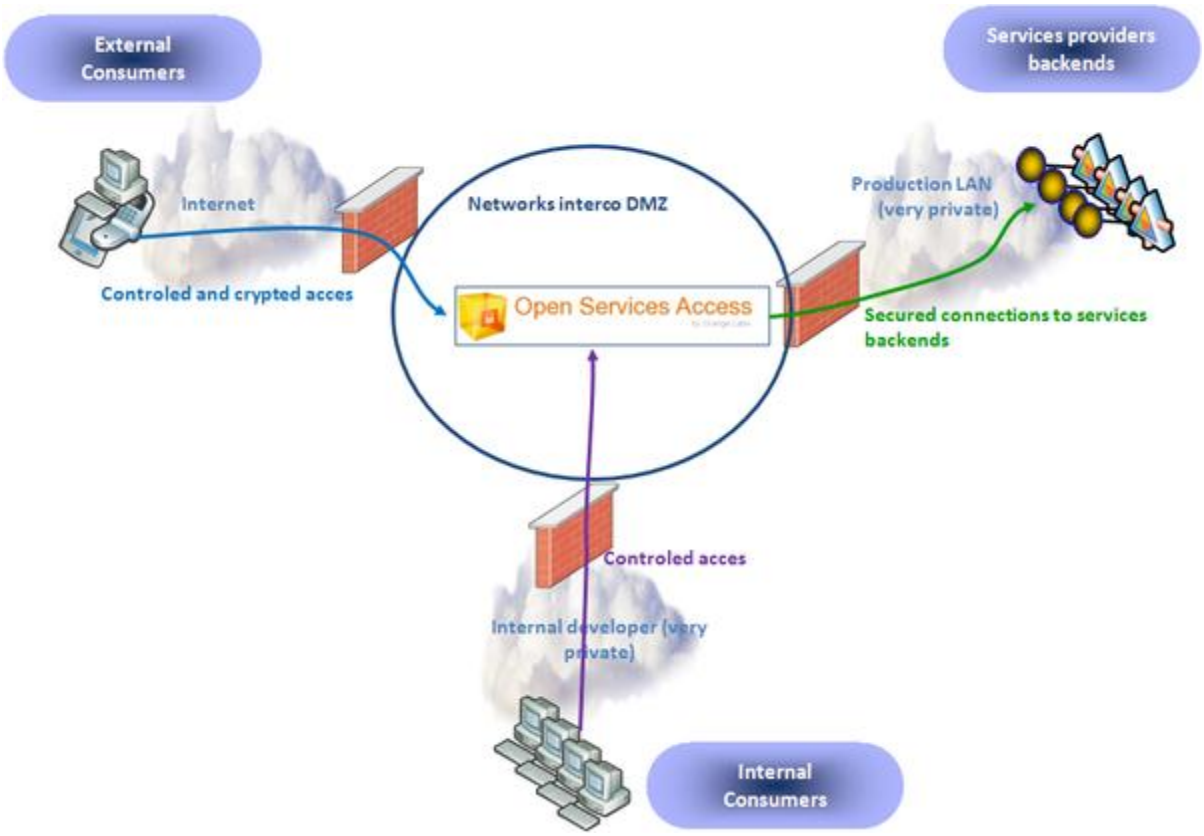
Of course, you want to avoid to re-implement or re-study for each one security enforcement and network connectivity solutions…… Let's imagine that your network topology looks like:



It's were OSA take place! By deploying OSA in your interco DMZ you are now able to:

- Give access to your services to external consumers through HTTPS
- Give access to your services to your developers through HTTP for debug
- Protected your services
- Manage network connectivity issues to private production network for only a single infrastructure (OSA)

Your network may become:



9.3.2 Ports in use after installation

- HTTP API Usage virtualhost configuration (reverse proxy): port 81
- HTTPS API Usage virtualhost configuration (reverse proxy): port 8443
- HTTPS Appliance management virtualhost configuration (reverse proxy): port 6443
- HTTP PHP Application hosting on localhost: port 82

This port is accessible only from a local client, and is not protected by authentication.

9.3.3 Web Services

The administration features are accessible throught WebServices, but this part is not documented. You can find the objects definition in the following xsd:

```
<xs:schema     xmlns:xs="http://www.w3.org/2001/XMLSchema"
               xmlns:ns1="http://nursery.orange.com/appliance/V1"
               targetNamespace="http://nursery.orange.com/appliance/V1"
               elementFormDefault="qualified"
               attributeFormDefault="unqualified">
               <xs:simpleType name="booleanWithInt">
```

```
              <xs:restriction base="xs:integer">
               <xs:enumeration value="0"/>
               <xs:enumeration value="1"/>
              </xs:restriction>
       </xs:simpleType>
       <xs:complexType name="ServicesType">
              <xs:sequence maxOccurs="unbounded">
                     <xs:element name="Service" type="ns1:ServiceType"
nillable="false" minOccurs="0"/>
              </xs:sequence>
       </xs:complexType>
       <xs:complexType name="ServiceType">
              <xs:sequence>
                     <xs:element name="Uri" type="xs:anyURI"
nillable="false"/>
                     <xs:element name="GroupUri" type="xs:string"
nillable="true"/>
                     <xs:element name="ServiceName" type="xs:string"
nillable="false"/>
                     <xs:element name="GroupName" type="xs:string"
nillable="true"/>
                     <xs:element name="IsIdentityForwardingEnabled"
type="ns1:booleanWithInt" nillable="true"/>
                     <xs:element name="IsGlobalQuotasEnabled"
type="ns1:booleanWithInt" nillable="true"/>
                     <xs:element name="IsUserQuotasEnabled"
type="ns1:booleanWithInt" nillable="true"/>
                     <xs:element name="IsPublished"
type="ns1:booleanWithInt" nillable="false"/>
                     <xs:element name="ReqSec" type="xs:unsignedLong"
nillable="true"/>
                     <xs:element name="ReqDay" type="xs:unsignedLong"
nillable="true"/>
```

```
                        <xs:element name="ReqMonth" type="xs:unsignedLong"
nillable="true"/>

                        <xs:element name="FrontEndEndPoint" type="xs:anyURI"
nillable="false"/>

                        <xs:element name="BackEndEndPoint" type="xs:anyURI"
nillable="false"/>

                        <xs:element name="BackEndUsername" type="xs:string"
nillable="true"/>

                        <xs:element name="BackEndPassword" type="xs:string"
nillable="true"/>

                        <xs:element name="IsHitLoggingEnabled"
type="ns1:booleanWithInt" nillable="true"/>

                        <xs:element name="IsUserAuthenticationEnabled"
type="ns1:booleanWithInt" nillable="true"/>

                        <xs:element name="AdditionalConfiguration"
type="xs:string" nillable="true"/>

                </xs:sequence>

        </xs:complexType>

        <xs:complexType name="GroupsType">

                <xs:sequence maxOccurs="unbounded">

                        <xs:element name="Group" type="ns1:GroupType"
nillable="false" minOccurs="0"/>

                </xs:sequence>

        </xs:complexType>

        <xs:complexType name="GroupType">

                <xs:sequence>

                        <xs:element name="Uri" type="xs:anyURI"
nillable="false"/>

                        <xs:element name="GroupName" type="xs:string"
nillable="false"/>

                        <xs:element name="Description" type="xs:string"
nillable="true"/>

                </xs:sequence>

        </xs:complexType>
```

```
	<xs:complexType name="UsersType">
		<xs:sequence maxOccurs="unbounded">
			<xs:element name="User" type="ns1:UserType"
nillable="false" minOccurs="0"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="UserType">
		<xs:sequence>
			<xs:element name="Uri" type="xs:anyURI"
nillable="false"/>
			<xs:element name="UserName" type="xs:string"
nillable="false"/>
			<xs:element name="Password" type="xs:string"
nillable="false"/>
			<xs:element name="Firstname" type="xs:string"
nillable="false"/>
			<xs:element name="Lastname" type="xs:string"
nillable="false"/>
			<xs:element name="Entity" type="xs:string"
nillable="false"/>
			<xs:element name="EmailAddress" type="xs:string"
nillable="true"/>
			<xs:element name="EndDate" type="xs:dateTime"
nillable="true"/>
		</xs:sequence>
	</xs:complexType>

	<xs:complexType name="UserQuotasType">
		<xs:sequence maxOccurs="unbounded">
			<xs:element name="UserQuota"
type="ns1:UserQuotaType" nillable="false" minOccurs="0"/>
		</xs:sequence>
```

```
	</xs:complexType>

	<xs:complexType name="UserQuotaType">

		<xs:sequence>

			<xs:element name="Uri" type="xs:anyURI"
nillable="false"/>

			<xs:element name="ServiceName" type="xs:string"
nillable="false"/>

			<xs:element name="ServiceUri" type="xs:anyURI"
nillable="true"/>

			<xs:element name="UserName" type="xs:string"
nillable="true"/>

			<xs:element name="UserUri" type="xs:string"
nillable="true"/>

			<xs:element name="ReqSec" type="xs:unsignedLong"
nillable="true"/>

			<xs:element name="ReqDay" type="xs:unsignedLong"
nillable="true"/>

			<xs:element name="ReqMonth" type="xs:unsignedLong"
nillable="true"/>

		</xs:sequence>

	</xs:complexType>




	<xs:complexType name="CountersType">

		<xs:sequence maxOccurs="unbounded">

			<xs:element name="Counter" type="ns1:CounterType"
nillable="false" minOccurs="0"/>

		</xs:sequence>

	</xs:complexType>

	<xs:complexType name="CounterType">

		<xs:sequence>

			<xs:element name="Uri" type="xs:anyURI"
nillable="false"/>
```

```
                        <xs:element name="UserName" type="xs:string"
nillable="true"/>

                        <xs:element name="ResourceName" type="xs:string"
nillable="false"/>

                        <xs:element name="TimeUnit" type="xs:string"
nillable="false"/>

                        <xs:element name="TimeValue" type="xs:dateTime"
nillable="false"/>

                        <xs:element name="Value" type="xs:unsignedLong"
nillable="false"/>

                </xs:sequence>

        </xs:complexType>

        <xs:complexType name="LogsType">

                <xs:sequence maxOccurs="unbounded">

                        <xs:element name="Log" type="ns1:LogType"
nillable="false" minOccurs="0"/>

                </xs:sequence>

        </xs:complexType>

        <xs:complexType name="LogType">

                <xs:sequence>

                        <xs:element name="Uri" type="xs:anyURI"
nillable="false"/>

                        <xs:element name="UserName" type="xs:string"
nillable="true"/>

                        <xs:element name="ServiceName" type="xs:string"
nillable="false"/>

                        <xs:element name="FrontEndUri" type="xs:string"
nillable="false"/>

                        <xs:element name="TimeStamp" type="xs:dateTime"
nillable="false"/>

                        <xs:element name="Status" type="xs:unsignedLong"
nillable="false"/>

                        <xs:element name="Message" type="xs:string"
nillable="true"/>
```

```
            </xs:sequence>

      </xs:complexType>


      <xs:complexType name="CounterExcedeedType">

            <xs:complexContent>

                  <xs:extension base="ns1:CounterType">

                        <xs:sequence>

                              <xs:element name="maxValue"
type="xs:integer"  nillable="false"/>

                        </xs:sequence>

                  </xs:extension>

            </xs:complexContent>

      </xs:complexType>

      <xs:complexType name="PaginatedLogsListType">

           <xs:sequence>

               <xs:element name="Length" type="xs:integer"/>

               <xs:element name="Previous" type="xs:anyURI"
nillable="true"/>

                           <xs:element name="Logs" type="ns1:LogType"
nillable="false" minOccurs="0" maxOccurs="unbounded"/>

              <xs:element name="Next" type="xs:anyURI"
nillable="true"/>

          </xs:sequence>

    </xs:complexType>

      <xs:complexType name="ErrorType">

            <xs:sequence>

                  <xs:element name="Code" type="xs:integer"
nillable="false"/>

                  <xs:element name="Label" type="xs:string"
nillable="false"/>

            </xs:sequence>
```

```
        </xs:complexType>


        <xs:element name="Service" type="ns1:ServiceType"/>

        <xs:element name="Services" type="ns1:ServicesType"/>


        <xs:element name="Group" type="ns1:GroupType"/>

        <xs:element name="Groups" type="ns1:GroupsType"/>


        <xs:element name="User" type="ns1:UserType"/>

        <xs:element name="Users" type="ns1:UsersType"/>


        <xs:element name="UserQuota" type="ns1:UserQuotaType"/>

        <xs:element name="UserQuotas" type="ns1:UserQuotasType"/>


        <xs:element name="Counter" type="ns1:CounterType"/>

        <xs:element name="Counters" type="ns1:CountersType"/>


        <xs:element name="Log" type="ns1:LogType"/>

        <xs:element name="Logs" type="ns1:LogsType"/>


        <xs:element name="Error" type="ns1:ErrorType"/>


        <xs:element name="PaginantedLogsList"
type="ns1:PaginatedLogsListType"/>

 </xs:schema>
```

10 S3C GE - Telecom AS - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

10.1 Introduction

The Telecom AS implements the a set of APIs that are to be developed on the Voice Call Control RESTful API. It allows to manage outgoing calls from a service logic via a REST interface. So, it is possible :

- to create a call between participants,
- to get information about a call (is the call terminated?),
- to get information about a call's participant,
- to add a participant to a call,
- to hang up a participant,
- to delete a call,
- to receive notifications about a participant according criteria as when the participant answers or as when the participant is busy,
- to play an audio message to participant,
- to be notified of any keypad interaction.

10.2 User Guide

As Telecom AS is an enabler, usage is dedicated to the programmers, so the User Guide section is not applicable.

10.3 Programmers guide

This section gives an overview of the different possibilities of use of the Telecom AS enabler. The samples of code follow the specifications of the enabler (Telecom_AS_API_Specification) based one the GSMA OneAPI standard (GSMA OneAPI V2 Specifications).

10.3.1 Samples of enabler using of mainly requests

10.3.1.1 *Create call*

POST /exampleAPI/thirdpartycall/callSessions HTTP/1.1

```
Content-Type: application/x-www-form-urlencoded

Content-Length: nnnn
```

```
Accept: application/json

{"callSessionInformation": {

"clientCorrelator": "104567",

"participant": [

{ "participantAddress": "tel:+4912345678901",

"participantName": "Max Muster"},

{"participantAddress": "tel:+4412345678901",

"participantName": "Peter E. Xample"

}

]

}}
```

The previous JSON request answers to create a call composed of two participants.

10.3.1.2 *Add a participant*

POST /exampleAPI/thirdpartycall/callSessions/cs002/participants HTTP/1.1

```
Content-Type: application/json

Accept: application/json

Content-Length: nnnn

Host: example.com:80

{"callParticipantInformation": {

"clientCorrelator": "224567",

"participantAddress": "tel:+1567890123456",

"participantName": "John E. Xample"

}}
```

The previous JSON request allows to add a participant to a call in which its id is « cs002 »

10.3.1.3 *Play an audio message to participants*

POST /exampleAPI/audiocall/messages/audio HTTP/1.1

```
Accept: application/xml
```

```
Content-Length: nnnn

Content-Type: application/x-www-form-urlencoded

Host: example.com

callSessionIdentifier=B45678&

callParticipant= tel%3A%2B4912345678901&

callParticipant= tel%3A%2B4412345678901&

mediaUrl=http%3A%2F%2Fwww.example.com%2Fann1.mp3&

mediaType=audio%2Fmpeg&

clientCorrelator=22345
```

In the URL request, it is answered to play to two participants an audio message located to « mediaURL ».

10.3.1.4 *Play an audio message to collect user interactions with it*

POST /exampleAPI/audiocall/interactions/collection HTTP/1.1

```
Accept: application/json

Content-Type: application/json

Content-Length: nnnn

Host: example.com

{"digitCapture": {

"callParticipant": "tel:+4912345678901",

"callSessionIdentifier": "F14567",

"clientCorrelator": "62345",

"digitConfiguration": {

"interruptMedia": "false",

"maxDigits": "1",

"minDigits": "1"

},

"playingConfiguration": {

"interruptMedia": "false",

"mediaType": "audio/mpeg",
```

```
"messageFormat": "Audio",

"playFileLocation":

"http://www.example.com/msg1.mp3"

}

}}
```

This JSON request allows to collect user interactions while an audio message is playing. In this example, The first digit is notified to service logic only. Moreover, the « interruptMedia » attributs set up the audio message is play in loop and it is not interrupted when user interacts.

10.3.2 Constraint of usage

For outgoing calls, the opening of a numbering plan should be asked explicitly to Orange. It is limited in time and number of calls. Any request is to be forwarded to the contact mentioned in the FI-WARE S3C Catalogue page.

11 S3C GE - Network Identity Management - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

11.1 Introduction

The NIM enables two functionalities within FI-WARE (see the Open Specification page for details). The features are the DeviceCapabilities, which is specified in GSMA OneAPI Device Capabilities (http://oneapi.aepona.com/f/files/OneAPI_V2_0_Device_Capabilities_REST.pdf) and Third Party Call Control, specified in GSMA OneAPI Voice Call Control (http://oneapi.aepona.com/f/files/OneAPI_V2_0_VoiceCallControl_REST.pdf). Since the NIM is an Application Server for providing additional functionalities of the network, there will be only the programmers guide for the services to utilize this function. There is no graphical user interface etc. to interact directly with the end user.

11.2 Users Guide

The Network Identity Management has no interface for a user to interact with. Therefore the Users Guide is not applicable here.

11.3 Programmers Guide

11.3.1 Preconditions

11.3.1.1 *Allow Subscribing*

The Network Identity Management needs to be declared as a secure service inside the IMS domain. Therefore the SIP Service IP address and SIP Port need to be incorporated into the service provisioning of the IMS core. The service shall be able to SUBSCRIBE any registered IMS subscriber to receive the device identifications. The NIM was tested with the OpenIMS Core implementation (see SCSCF Documentation of OpenIMS Core). Inside the configuration of the S-CSCF the route[SUBSCRIBE] element was enhanced with the following ISC_from_AS-condition:

```
If(S_can_subscribe() || ISC_from_AS("orig")) {

//original content

} else {

//original content
```

}

11.3.1.2 *Check GRUU based Requests*

To distinguish between device-id routing and normal IMS behaviour with request branching, the following patch needs to be included into the SCSCF of OpenIMS-module :

```
Index: registrar.c
===================================================================
--- registrar.c      (revision 1182)
+++ registrar.c      (working copy)
@@ -1451,7 +1451,11 @@
                              set_ruri_q(c->qvalue);


                              ret = CSCF_RETURN_TRUE;
-                             if (append_branches){
+
+                             /*
+                              * Check if GRUU was sent, ignore append
branches
+                              */
+                             if (append_branches && !pub_gruu.len > 0){
                                     c = c->next;
                                     while(c){
                                            if (r_valid_contact(c))
```

11.3.2 Device Capabilities

To come up with the use case example “various end terminals for one subscriber” the REST based interface Device Capabilities is not completely feasible for use. The specification assumes only device per given subscriber identity. Therefore the syntax is enhanced to cope different unique devices per subscriber. It is managed with the setup of a JSON array where the original syntax of the Device Capabilities is included within each array element. To get used to it, follow the example: Request:

```
POST /oneapi/devicecapabilities/alice@snc-ims.tlabs.de/capabilities
HTTP/1.1
Host: 192.168.7.10:8080
```

Response:

```
HTTP/1.1 201 Created

Content-Type: application/json

Date: Tue, 18 Jun 2013 14:00:19 GMT

Transfer-Encoding: chunked


Created subscription for alice@snc-ims.tlabs.de
```

The POST message of the given Subscriber creates a watcher towards the network for all registered devices. After a grace time (~ 30s) the GET request

```
GET /oneapi/devicecapabilities/alice@snc-ims.tlabs.de/capabilities
HTTP/1.1

Host: 192.168.7.10:8080

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:21.0)
Gecko/20100101 Firefox/21.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: null

Accept-Encoding: gzip, deflate

Connection: keep-alive
```

will send out

```
HTTP/1.1 200 OK

Content-Type: application/json

Date: Tue, 18 Jun 2013 14:29:32 GMT

Transfer-Encoding: chunked


{

 "resourceURL":
"http://192.168.7.10:8080/oneapi/devicecapabilities/alice@snc-
ims.tlabs.de/capabilities",

 "deviceList": [

   {

     "expires": 3573,
```

```
      "name": "alice@snc-ims.tlabs.de;gr=f81d4fae-7dec-11d0-a765-
00a0c91e6bf6",

      "deviceId": "f81d4fae-7dec-11d0-a765-00a0c91e6bf6"

    },

    {

      "expires": 3572,

      "name": "alice@snc-ims.tlabs.de;gr=ac235eae-efb1-1843-a21a-
84120caef246",

      "deviceId": "ac235eae-efb1-1843-a21a-84120caef246"

    }

  ]
```

}

The new array element “deviceList” was added to fulfil the requirement of various devices per subscriber. The “expires” value (in seconds) means how long the registration of this instance is still valid. The key deviceId shows the unique instance ID - in this example it is simplified. The key “name” is the public identification to reach a specific device of the subscriber.

In release 2, this is an 1:1 mapping from the real instance ID to the public identification.

11.3.3 Third Party Call with device specific addressing

Next to identifying every logged in device per subscriber, a service is allowed to create a third party call between two end points in an IMS based network. In this case, the third party is the Network Identity Management which is triggered by the HTTP service request:

```
POST http://192.168.7.10:8080/oneapi/callSessions/

Host: 192.168.7.10:8080

Accept: application/json

Content-Type: application/json


{

 "callSessionInformation": {

  "clientCorrelator": "304567",

    "participant": [
```

```
      {
        "participantAddress":"sip:bob@snc-ims.tlabs.de",
        "participantName":"Bob"
      },
      {
        "participantAddress":"sip:alice@snc-ims.tlabs.de",
        "participantName":"Alice"
        "participantInstance": "f81d4fae-7dec-11d0-a765-00a0c91e6bf6"
      }
    ],
  "participantAnnouncement": "predefinedAnnouncement1ForParticipant"
 }
}
```

It is important to know, that the attribute "participantInstance" is one of the device identifications requested via the device capabilities. This attribute is not standardized, but allows to address a specific end-terminal instead of all entities of a subscriber.

The response is a 201 created HTTP message with a resource Link to get the status of the current created call:

```
201 Created
Content-Type: application/json
Date: Wed, 19 Jun 2013 08:49:23 GMT
Transfer-Encoding: chunked

{
  "resourceReference": {
    "resourceURL":
"http://192.168.7.10:8080/oneapi/callSessions/0813217086fe6725a61443c481841b06@192.168.7.10"
  }
}
```

12 S3C GE - Seamless Network Connectivity - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

12.1 Introduction

This manual contains information about the use and programming of the S3C's Seamless Network Connectivity functionality available through the FI-WARE Generic Enabler implementation. The functionality is described in the S3C Open Specification page. The reader of this manual should also be familiar with the contents of the Installation & Administration manual.

This User Guide gives an overview of the flow setup. It explains the pre installed rules by the GE installation and provides information about how to list, add, replace and delete new application flows. Furthermore a list of common matching patterns is provided.

This release allows manual configuration only. There is no API for doing that in a programmatic way (Therefore there is no programmer manual available).

12.2 User Guide

12.2.1 Flow setup

If you want to add, change or delete the rules provided as example rules, you can do this by typing in the commands manually or write them down in the /etc/rc.local.snc script. The second option has the advantage that after a reboot they will automatically be configured. Normally you should do both, using the command for testing and writing them down in the file if you figured out that they are working.

CAUTION: It is possible to build a loop. That will cause much traffic and a high load on both machines.

12.2.1.1 *Explanation of sample rules on SCC*

Example Rules SCC

Nr.	Rule	Explanation
1	in_port=LOCAL,action=output:1	If something is going in from the local bridge interface, then it will immediately outputted on port 1 (UPPER_TUNNEL)
2	in_port=1,action=LOCAL	Everything which is ingress traffic from the upper tunnel, it will be outputted on the bridge interface

3	in_port=2,action=LOCAL	Exactly like the rule above, but now it is ingress traffic from the lower tunnel
4	priority=40000,in_port=LOCAL,tcp,tp_dst=5002,action=output:2	TCP traffic from client applications with the destination port 5002 will be send through the lower tunnel. Even though we already have a rule that is doing nearly the same (rule 1), because of the priority. The normal priority is 32768. The higher the value, the higher the priority.
5	priority=40000,in_port=LOCAL,tcp,tp_dst=6666,action=output:2	Same as above
6	priority=40000,in_port=LOCAL,tcp,tp_dst=80,action=output:2	Again, it is the same as above, but in the Explanation of the SCS example Rules we will see the difference.

12.2.1.2 *Explanation of sample rules on SCS*

Example Rules SCS

Nr.	Rule	Explanation
1	in_port=LOCAL,action=output:1	If something is going in from the local bridge interface, then it will immediately outputted on port 1 (UPPER_TUNNEL)
2	in_port=1,action=LOCAL	Everything which is ingress traffic from the upper tunnel, it will be outputted on the bridge interface
3	in_port=2,action=LOCAL	Exactly like the rule above, but now it is ingress traffic
4	priority=40000,in_port=LOCAL,tcp,tp_src=5002,action=output:2	TCP traffic from client applications with the source port 5002 will be send through the lower tunnel. Even though we already have a rule that is doing nearly the same (rule 1), because of the priority. The normal priority is 32768. The higher the value, the higher the

		priority.
5	priority=40000,in_port=LOCAL,tcp,tp_src=6666,action=output:2	Same as above
6	priority=40000,in_port=LOCAL,tcp,tp_src=80,action=output:1	Instead of using the lower tunnel like the SCC, it is now using the upper tunnel. (Reasons could be: Downlink of the upper tunnel is much faster than the lower tunnel)

12.2.1.3 *List flows*

```
ovs-ofctl dump-flows <Bridge Interface>
```

12.2.1.4 *Add flows*

```
ovs-ofctl add-flow <Bridge Interface> <FLOW>
```

12.2.1.5 *Change flows*

```
ovs-ofctl mod-flows <Bridge Interface> <FLOW>
```

12.2.1.6 *Delete flows*

Delete all flows:

```
ovs-ofctl del-flows <Bridge Interface>
```

Delete a single flow:

```
ovs-ofctl del-flows <Bridge Interface> <FLOW>
```

12.2.1.7 *Flow syntax*

Here some matching fields for the flows:

- in_port
- dl_vlan
- dl_src
- dl_dst
- dl_type

- nw_src
- nw_dst
- nw_proto
- nw_tos
- nw_ttl
- tp_src (please make sure that you also added tcp or udp in the flow rule)
- tp_dst (please make sure that you also added tcp or udp in the flow rule)
- ipv6_src
- ipv6_dst
- …

Please feel free to look at the following link for more details about the flow syntax [1].

12.3 Programmers Guide

The Seamless Network Connectivity does not require specific programming to be used, therefore the Programmers Guide is not applicable here.

13 S3C GE - EPC OTT API - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

13.1 Introduction

This User and Programmers Guide relates to the Service Capability, Connectivity and Control GE, which is part of the I2ND chapter, in particular, it refers to the EPC-OTT component. You can find more information about the EPC-OTT API in its Open Specification.

The EPC-OTT API enables the application developers to use a RESTful interface within their application to provide their indications towards the operator core network. The following section describes how the application developers can influence the communication through the operator network as well as through the specific radio access networks.

The following features are available from the core network perspective:

- Reservation of resources for the specific communication sessions and for each of the sesssion's data flows
- Modification of the previously reserved resources
- Termination of the previously reserved resources
- Event notifications in case the status of the connectivity of the mobile devices changes

These features are provided through the Rx Diameter interface. The EPC-OTT API is providing a translation between the Diameter interface towards the operator core network and the RESTful interface towards the applications.

Additionally to these features, the EPC-OTT API provides through the RESTful API the applications with information on the access networks on the vicinity of the mobile devices and a means to transmit an indication on which access network the communication of the application should continue on. As multiple applications run in parallel and as the final decision on which access network is most suitable to support the device communication is pertaining to the operator, the indication from the application may be ignored.

An implicit functionality is offered directly by the core network: the capacity to transmit IP messages to the end devices in an appropriate manner including small messages or cached messages.

13.1.1.1 *Restrictions*

The EPC-OTT API component is delivered in a testbed demonstration version and is bound to a limited functionality.

Additionally, the operator core network functionality was not designed to be exposed even to applications for which a business relationship with the operator exists. Therefore, the EPC-OTT exposure of the operator core network is restricted also by the exposure features offered by the underlying operator infrastructure.

The following limitations are considered:

- The QoS resource reservations, modifications, and termination are bound to the International Mobile Subscriber Identifier (IMSI) and not to other possible identities
- The QoS resource reservations are not actually realized over the radio access network. Instead a gating and data shaping is realizing the specific features in the core network.
- The access network selection allows only for access network type selection and not for the selection of the specific access networks in the vicinity of the device. A finer granularity requires a higher exposure of the internals of the operator network, which may pose a security threat.
- The operator network handles the data traffic handling without exposing an API, as good as possible in the specific conditions, thus handling small size messages automatically, without requiring additional API functionality.

13.2 User Guide

The OTT EG does not provide a user interface per se. It is meant to be used as a REST server, so the typical user would be a user using a REST client or a programmer implementing the REST interface. Therefore the usage of the GE is explained in the next section.

13.3 Programmer Guide

This part of the User and Programmers Guide explains how programmers are intended to use the EPC OTT API. THe OTT GE provides a REST interface, so any application developer who wants to make use of it, need to implement or reuse a REST client. The specification of the data objetcs can be found in the Open Specification section of the OTT GE.
In the following, a few example requests are explained with some more or less useful but valid data.

13.3.1 Starting a QoS Session

In order to reserve resources for a specific connection, an HTTP POST request must be send to the OTT API server. The URL must contain the username (due to the above mentioned restriction given as IMSI number):

```
http://ip:port/ngsi.applicationDrivenQoS/rest/1/QoSManager/startSession?authkey=abcd&userName=001011234567890
```

The content of the POST request must contain an XML file with the parameters of the connection:

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<qoSFeatureProperties>
    <duration>
```

```
        <units>100000</units>
    </duration>
    <upStreamSpeedRate>300000</upStreamSpeedRate>
    <downStreamSpeedRate>300000</downStreamSpeedRate>
    <otherProperties>
        <name>Source_ip</name>
        <value>192.168.1.30</value>
    </otherProperties>
    <otherProperties>
        <name>Source_port</name>
        <value>22</value>
    </otherProperties>
    <otherProperties>
        <name>Destination_ip</name>
        <value>192.168.3.33</value>
    </otherProperties>
    <otherProperties>
        <name>Destination_port</name>
        <value>1234</value>
    </otherProperties>
    <otherProperties>
        <name>Protocol</name>
        <value>udp</value>
    </otherProperties>
    <otherProperties>
        <name>MediaType</name>
        <value>DATA</value>
    </otherProperties>
    <otherProperties>
```

```
        <name>Framed_IP</name>
        <value>192.168.3.33</value>
    </otherProperties>
    <otherProperties>
        <name>Direction</name>
        <value>out</value>
    </otherProperties>
</qoSFeatureProperties>
```

The response contains a string which is the session id of the newly registered session. This session id needs to be used for the following requests.

13.3.2 Modification of Reserved Resources

In order to modify the QoS parameter of the created session, an HTTP POST request must be sent to

```
http://ip:port/ngsi.applicationDrivenQoS/rest/1/QoSManager/modifySession?
authkey=abcd&mediaIdentifier=1&sessionKey=1234
```

with an XML containing the new values of the new resource requirements:

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<qoSFeatureProperties>
    <upStreamSpeedRate>6000</upStreamSpeedRate>
    <downStreamSpeedRate>6000</downStreamSpeedRate>
</qoSFeatureProperties>
```

The sessionKey parameter of the URL is the session ID that was returned when the flow has been registered with registerFlow.

13.3.3 Ending a QoS Session

To release the reserved resources, an HTTP GET request must be send to the OTT API server containing the session ID:

```
http://ip:port/ngsi.applicationDrivenQoS/rest/1/QoSManager/endSession?aut
hkey=abcd&sessionKey=1234
```

13.3.4 Getting the QoS Status of a User

In order to get the resource reservation status of a user, an HTTP GET request must be send to the OTT API server containing the user name whose reservation shall be fetched:

```
http://ip:port/ngsi.applicationDrivenQoS/rest/1/QoSManager/getQoSStatus?a
uthkey=abcd&userName=001011234567890
```

The response contains an XML file with all the QoS parameters of the requested QoS sessions.

13.3.5 Subscribing for Event Notfications

```
http://ip:port/ngsi.applicationDrivenQoS/rest/1/QoSManager/endSession?aut
hkey=abcd&userName=001011234567890&event=INDICATION_OF_RELEASE_OF_BEARER
```

The following events are possible:

```
*INDICATION_OF_RELEASE_OF_BEARER

*INDICATION_OF_LIMITED_PCC_DEPLOYMENT

*INDICATION_OF_FAILED_RESOURCE_ALLOCATION

*INDICATION_OF_OUT_OF_CREDIT

*INDICATION_OF_SUCCESSFUL_RESOURCE_ALLOCATION
```

For more information see the Open API Secification

13.3.6 Transmission of an IP message

The transmission of an IP message is implicit to the API being it a small size message or a set of small size messages which are cached in the operator network.

14 S3C GE - Network Positioning Enabler - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

14.1 S3C GE - Network Positioning Enabler - User and Programmers Guide

14.1.1 Introduction

This manual contains information about the use and programming of the S3C's Positioning Enabler. The functionality is described in the Network Positioning Enabler Open API Specification. The reader of this manual should also be familiar with the contents of the Installation & Administration manual.

The User and Programmers Guide is used to provide an insight for the Positioning Enabler users as well as developers. From one side, it describes the functionalities provided to the user as well as the available possible ways for interaction with the Positioning Enabler. It introduces the user's Android client as well as the 3rd party interface used to create network-specific messages. On the other hand, it gives an overview to programmers (developers) about the functionality of the Positioning Enabler and its interaction with the mobile client as well as with the mobile core network.

14.1.1.1 ***Restrictions***

The Positioning Enabler (PE) platform does only work with a mobile operator core network, e.g. the Evolved Packet Core (EPC), and hence requires an interface to the EPC. Within FI-WARE, the PE maintains an interface to the OpenEPC core network testbed provided by Fraunhofer FOKUS. For illustrating the functionality of the PE it is required that the OpenEPC has at least two radio access networks up and running. Furthermore, at least one user equipment (UE) has to be provisioned in the network as well as connected to the core network via a 3GPP or non-3GPP access network supported by the OpenEPC.

The EPC is delivered in a testbed demonstration version. The EPC consists of a single WiFi access network (non-3GPP) and 2 other radio access networks (3GPP), i.e. 2G and 4G. For a UE to connect to the EPC via these access networks an EPC-compatible SIM card is required. If you want to use the provided client app for testing, the EPC-compatible SIM card must have the IMSI "01011234567890". The EPC's access network selection allows only for access network type selection and not for the specific access networks in the vicinity of the device. Given this fact, the access network selection priorities are passed from the EPC to the UE, switching the access networks off would be required in order to illustrate the UE switching to the next access network in the list of priorities.

Ensuring that the SIM card with the above mentioned IMSI is connected to an EPC access network is inevitable for demonstrating the functionality of the Positioning Enabler.

Every UE connected to the EPC testbed has no Internet access and is thus not reachable via the Positioning Enabler APIs.

14.2 User Guide

14.2.1 Client

The client needs to be installed on an Android device. If the client is installed successfuly it is accessible by starting the App called 'Fiware'.

14.2.1.1 *Logging in*

To log in click on the “Log in” button. You will then be redirected to Google’s openId page, where you have to provide your google account credentials.

14.2.1.2 *Logging out*

To log out click the “Logout” button in the menu.

14.2.1.3 *Subscribing to a service*

Via login the user is automatically subscribed to the "Fiware Service".

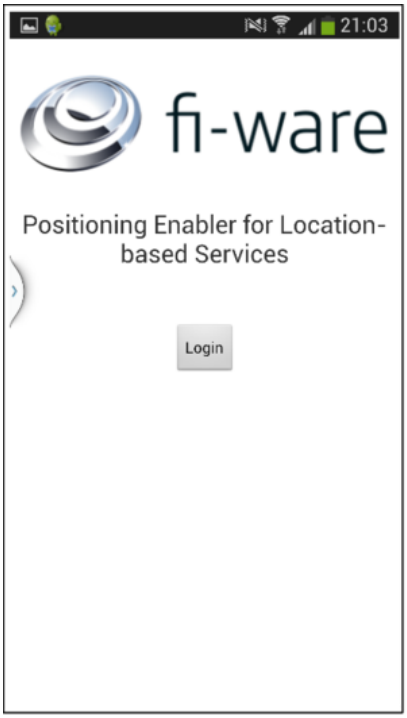
14.2.1.4 *Starting and ending the background tracking*

The user can subscribe for background tracking by clicking on the “Subscribe” button. The other way around the user can unsubscribe by clicking on the “Unsubscribe” button.

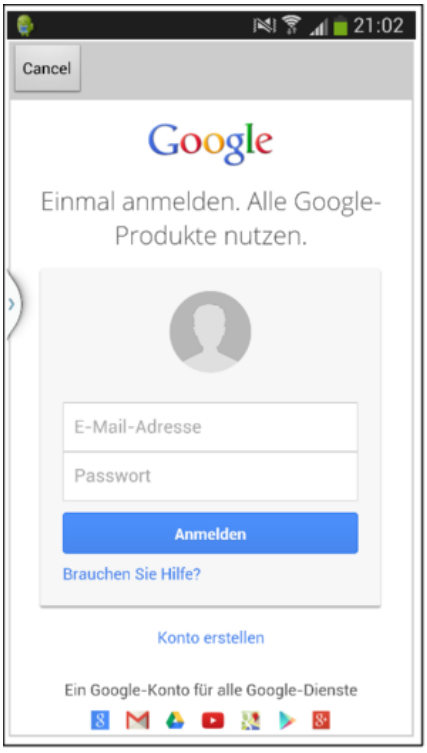
14.2.1.5 *Position request*

The user can also request the position of a UE by clicking on the “Get position” button. In this case a toast will be shown to the user with the location of the UE the user subscribed to.

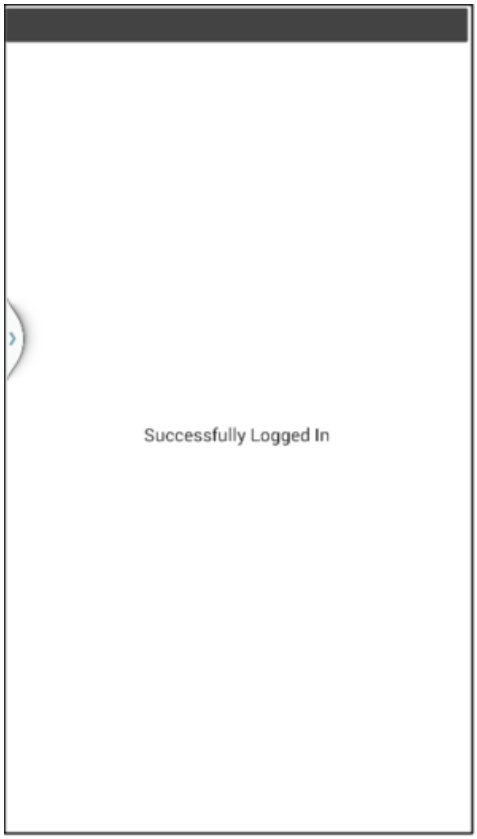
14.2.1.6 *Sample Screenshots of the Client*



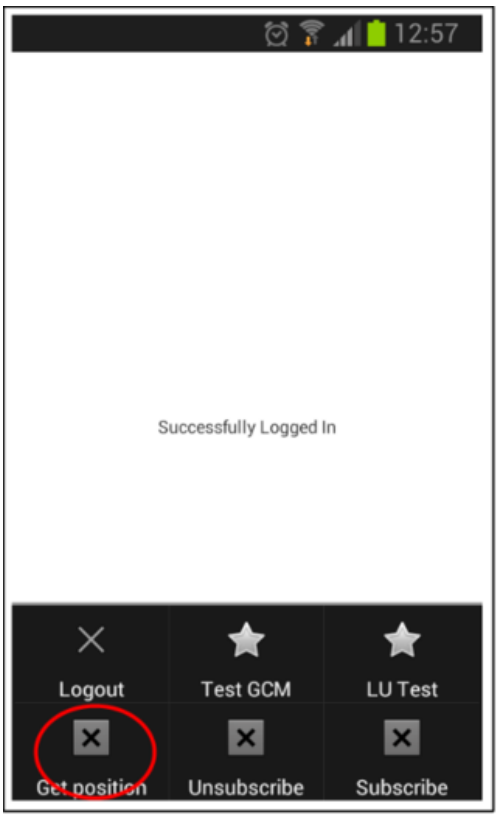
Start screen of the FI-WARE client app



Login screen of the the FI-WARE client app



Home screen of the FI-WARE client app



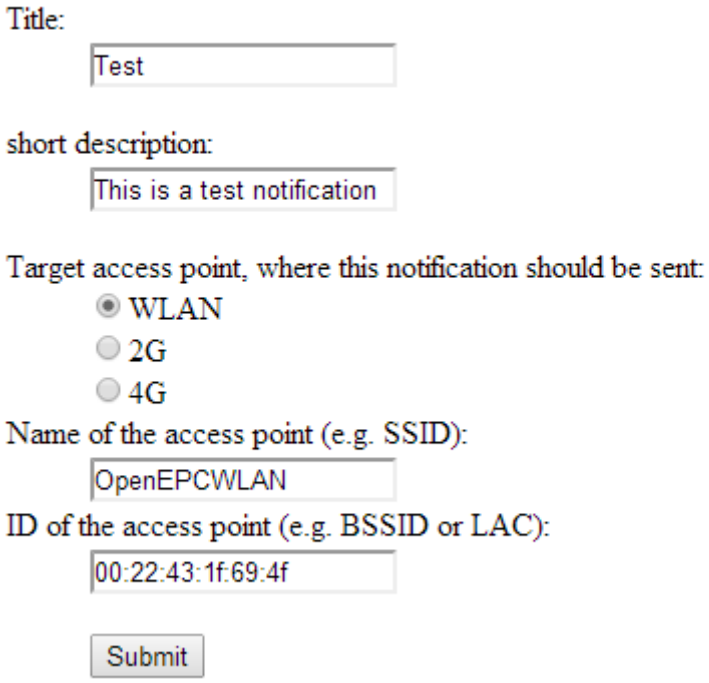
Testing tools screen

The position of the UE the user subscribed to is shown

14.2.2 3rd party interface

14.2.2.1 *Creation of a notification*

For 3rd parties it is possible to use the 3rd party interface to create new notifications which will be shown to users if they connect to the specified access point. Upon creation the server sends back the id of the new notification.



{peroot} = ip:port/PositioningEnabler/api

Access network	Identifier
WLAN	00:22:43:1f:69:4f (BSSID)
2G	0 (LAC)
4G	FFFE (LAC)

Request

```
POST /service/fiware/notification HTTP/1.1

Accept: */*

Accept-Charset: UTF-8

Content-Type: application/json; charset=UTF-8


{

 "title": "Test",

 "shortDescription": "This is a test notification",
```

```
 "targetAP": "3#OpenEPCWLAN#00:22:43:1f:69:4f"

}
```

Response

```
HTTP/1.1 201 Created

Content-Type: text/plain



{notificationid}
```

14.3 Programmers Guide

In the following sections this precondition holds:

- {peroot} = ip:port/PositioningEnabler/api

14.3.1 Registering and authenticating using OpenId

In the case of a new user logging in to create a new user account or an old user logging in again the programmer should use the same URL:

```
{peroot}/login/openid/provider/{provider}/{userId}
```

In the case of a new user {userId} should be null. The server redirects to the OpenId login page and then redirects the client to a URL starting with leanengine://. After a successful login the server responds with a userId (the same as mentioned in the URL if the user already existed in the database) and the auth token. If the login failed the server redirects the client to leanengine://error.

It is only possible to use Google as the OpenID provider.

Request

```
GET /login/openid/provider/{provider}/{userId}
```

Response • Successful login redirects to

```
leanengine://.../?auth_token={authToken}&id={userId}
```

• **Error on login** redirects to

```
leanengine://error
```

14.3.2 Logout

For logging a user out the programmer has to call the following URL with a HTTP DELETE method:

```
{peroot}/logout/openid/user/{userId}
```

Request

```
DELETE /logout/openid/{userid} HTTP/1.1

Basic {basicauthtoken}
```

Response

```
HTTP/1.1 200 OK
```

14.3.3 Getting the location of a user from the ANDSF

To get the current access network a user is connected to, connect to the MySQL database “andsf_db” (user = password = “andsf”) in the openEPC network and query the database with the IMSI of the user as follows:

```
SELECT location_xml FROM s14_clients

WHERE subscription_id_data = 001011234567890;
```

This query will return a XML file which is parsed by a SAX parser and can look like the following examples:

- In case it is connected to non-3GPP access network, i.e. WLAN in this example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<UE>

 <SubscriptionId>

  <Data>001011234567890</Data>

  <Type>1</Type>

 </SubscriptionId>

 <Name>OpenEPC-MM</Name>

 <DiscoveryInformation>

  <OpenEPCWLAN>

   <AccessNetworkType>3</AccessNetworkType>

   <AccessNetworkArea>

    <WLAN_Location>

     <OpenEPCWLAN>

      <SSID>OpenEPCWLAN</SSID>

      <BSSID>00:21:63:a4:49:f5</BSSID>

     </OpenEPCWLAN>

    </WLAN_Location>
```

```
    </AccessNetworkArea>
    <AccessNetworkInformationRef>OpenEPCWLAN</AccessNetworkInformationRef>
   </OpenEPCWLAN>
  </DiscoveryInformation>
  <UE_Location>
   <WLAN_Location>
    <OpenEPCWLAN>
     <SSID>OpenEPCWLAN</SSID>
     <BSSID>00:21:63:a4:49:f5</BSSID>
    </OpenEPCWLAN>
   </WLAN_Location>
  </UE_Location>
  <S14Alert>
   <AlertIP>192.168.103.54</AlertIP>
   <AlertPort>10005</AlertPort>
  </S14Alert>
</UE>
```

- In case it is connected to 3GPP access network, i.e LTE in this example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<UE>
  <SubscriptionId>
   <Data>001011234567890</Data>
   <Type>1</Type>
  </SubscriptionId>
  <Name>OpenEPC-MM</Name>
  <DiscoveryInformation>
   <OpenEPCWLAN>
    <AccessNetworkType>3</AccessNetworkType>
```

```
    <AccessNetworkArea>
     <WLAN_Location>
      <OpenEPCWLAN>
       <SSID>OpenEPCWLAN</SSID>
       <BSSID>00:22:43:1f:69:41</BSSID>
      </OpenEPCWLAN>
     </WLAN_Location>
    </AccessNetworkArea>
    <AccessNetworkInformationRef>OpenEPCWLAN</AccessNetworkInformationRef>
   </OpenEPCWLAN>
   <Test_PLMN_1-1>
    <AccessNetworkType>1</AccessNetworkType>
    <AccessNetworkInformationRef>Test1-1</AccessNetworkInformationRef>
   </Test_PLMN_1-1>
  </DiscoveryInformation>
  <UE_Location>
   <TGPP_Location>
    <Test1-1>
     <PLMN>TestPLMN1-1</PLMN>
     <LAC>FFFE</LAC>
     <EUTRA_CI>807</EUTRA_CI>
    </Test1-1>
   </TGPP_Location>
  </UE_Location>
  <S14Alert>
   <AlertIP>192.168.103.54</AlertIP>
   <AlertPort>10005</AlertPort>
  </S14Alert>
</UE>
```

The <Type> tag describes the identification type. '1' stands for identification by IMSI. Access networks between <DiscoveryInformation> tags show networks the UE can "see". Between <UE_Location> tags you can see the access network the UE is connected to. The differentiation between WLAN and 3GPP is made via the first tag after <UE_Location> (<WLAN_Location> vs <TGPP_Location>).

14.3.4 Subscription and Unsubscription

Subscription and unsubscription are realized via the same URL:

```
{peroot}/user/{userId}/subscription/{serviceName}
```

The difference is the HTTP method being used. To subscribe, the programmer has to use POST, to unsubscribe he has to use DELETE.

Request

```
POST /user/{userid}/subscription/{servicename} HTTP/1.1

Basic {basicauthtoken}

Accept: application/json

Accept-Charset: UTF-8

Content-Type: application/json

Host: {serverroot}


{

 "imsi": {imsi},

 "mobilePushId":{mobilePushId}

}
```

Response

```
HTTP/1.1 200 OK
```

14.3.5 Manipulation of a notification

Via the REST API it is also possible to delete, update or retrieve an already stored notification. All of them are realized with one URL:

```
{serverroot}/service/fiware/{notificationId}
```

In order to retrieve a notification, a user has to send a HTTP GET method to the URL. In order to update a notification or store a new one with a specific id, a user has to send a HTTP POST method to the URL, containing a JSON string representing a notification:

```
{
 "title": "New notification title",
 "shortDescription": "new short",
 "targetAP": "00:80:41:ae:fd:7e"
}
```

In order to delete a notification, a user has to send a HTTP DELETE method to the URL.

- **Successful deletion of a notification**

Request

```
DELETE /service/fiware/notification/740568d9-ee22-4483-8f96-c807eb03364a
HTTP/1.1
Accept: */*
Host: {serverroot}
```

Response

```
Request URL: {serverroot}/service/fiware/notification/740568d9-ee22-4483-
8f96-c807eb03364a
Request Method: DELETE
Status Code: 204 Resource updated
```

- **Successful retrieval of a notification**

Request

```
GET /service/fiware/notification/740568d9-ee22-4483-8f96-c807eb03364a
HTTP/1.1
Accept: application/json; charset=UTF-8
Host: {serverroot}
```

Response

```
Request URL: {serverroot}/service/fiware/notification/740568d9-ee22-4483-
8f96-c807eb03364a
Request Method: GET
Content-Type: application/json; charset=UTF-8
Status Code: 200 OK
{
 "title": "New notification title",
 "shortDescription": "new short",
```

```
 "targetAP": "00:80:41:ae:fd:7e"

}
```

- **Successful update of a notification** (this would create a notification with the given id, if it does not exist)

Request

```
POST /service/fiware/notification/740568d9-ee22-4483-8f96-c807eb03364a
HTTP/1.1

Accept: */*

Accept-Charset: UTF-8

Content-Type: application/json; charset=UTF-8

Host: {serverroot}

{

 "title": "New notification title",

 "shortDescription": "new short",

 "targetAP": "00:80:41:ae:fd:7e"

}
```

Response

```
Request URL:
{serverroot}/PositioningEnabler/api/service/fiware/notification/740568d9-
ee22-4483-8f96-c807eb03364a

Request Method: POST

Status Code: 204 Resource updated
```

14.3.6 Position request

A programmer can request the access point / access network a user is connected to via the following URL with a HTTP GET method:

```
{peroot}/user/{IMSI}/position
```

Request

```
GET /user/{imsi}/position HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
```

```
{
 "apKey": {apkey},
 "apName": {apname}
}
```

14.3.7 Sending a push notification

The programmer can manually use the Positioning Enabler API to send a notification to a specific user. For that he has to send a JSON string of this format

```
{
 "title": "notification title",
 "shortDescription": "short description",
}
```

to the following URL with a HTTP POST method:

```
{peroot}/pushnotification/{userId}
```

Request

```
POST /pushnotification/{userid} HTTP/1.1
Accept: */*
Accept-Charset: UTF-8
Content-Type: application/json; charset=UTF-8
Host: {serverroot}


{
 "title": "notification title",
 "shortDescription": "short description"
}
```

Response

```
HTTP/1.1 200 OK
```

UE Network Localization: The matching between the access network the UE is connected to and the target access point a 3rd party has specified, is realized in the following way:

1. Every subscription is bound to an access network, i.e. Target AP, which apparently specified through the 3rd party web interface.
2. If the UE is connected to an access network and has at least one subscription, the PE checks if the new connected access network is equal to one of the access networks the UE is subscribed to.
3. If this is the case, the user(Android Client) gets a notification that the UE is connected to this specific access network.